A Unifying Framework for Parallelizing Sequential Models with Linear Dynamical Systems

Anonymous authors
Paper under double-blind review

Abstract

Harnessing parallelism in seemingly sequential models is a central challenge for modern machine learning. Several approaches have been proposed for evaluating sequential processes in parallel using fixed-point methods, like Newton, Picard, and Jacobi iterations. In this work, we show that these methods can be understood within a common framework based on linear dynamical systems (LDSs), where different iteration schemes arise naturally as approximate linearizations of a nonlinear recursion. This unifying view highlights shared principles behind these techniques and clarifies when particular fixed-point methods are most likely to be effective. By bridging diverse algorithms through the language of LDSs, our framework provides a clearer theoretical foundation for parallelizing sequential models and points toward new opportunities for efficient and scalable computation.

1 Introduction

Sequential processes are ubiquitous in machine learning models. Evaluating a recurrent neural network (Goodfellow et al., 2016), sampling a diffusion model (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021b), generating from a deep state space model (Gu et al., 2022; Smith et al., 2023; Orvieto et al., 2023; Gu & Dao, 2024), and unrolling layers of a deep neural network (He et al., 2016; Vaswani et al., 2017) all involve sequential computations. Naively, these computations require time proportional to the length of the input or the depth of the architecture, and in some cases, they may not take full advantage of hardware accelerators like GPUs and TPUs (Lim et al., 2024). However, certain sequential computations — namely, *linear* recursions or linear dynamical systems (LDSs) — can be evaluated in parallel using techniques like the parallel scan (Blelloch, 1990; Fatahalian & Olukotun, 2024). Indeed, the parallelizability of linear recursions is key to efficiently evaluating many deep state space models (Smith et al., 2023; Gu & Dao, 2024). A natural question is whether other sequential processes in machine learning, which typically correspond to *nonlinear* recursions, can be similarly accelerated.

On first inspection, the parallel scan algorithm does not seem to generalize to nonlinear recursions. The reason it applies to linear recursions is that the composition of two linear functions remains linear, whereas the composition of two nonlinear functions is generally more complicated. For example, the composition of two quadratic functions is a quartic function. Nevertheless, recent works have proposed several techniques to parallelize nonlinear recursions, including Jacobi (Song et al., 2021a), Picard (Shih et al., 2023), and Newton (Danieli et al., 2023; Lim et al., 2024; Gonzalez et al., 2024) iterations. These techniques were originally applied to different machine learning problems and follow different notations and intuitions, which obscures their underlying similarities.

In this paper, we show that Jacobi, Picard, and Newton iterations all solve a fixed-point problem by iteratively linearizing the nonlinear recurrence and evaluating the resulting linear dynamical system with a parallel scan. In addition to providing a unifying perspective on these methods, we discuss the properties of each approach and their applicability for different types of problems. While the general connections between Picard and Newton iterations and their convergence rates for solving nonlinear equations have long been known by the applied mathematics community (Ortega & Rheinboldt, 1970), the novel contribution of this work is emphasizing the tight connection between these approaches for the specific problem of evaluating a nonlinear

recursion, a broadly important problem in machine learning. In particular, we show how each fixed-point iteration reduces to evaluating an LDS, which crucially allows the use of parallel computation in what at first seems to be an inherently sequential problem.

Our contributions include the following:

- In Section 2, we show how Newton, quasi-Newton, Picard, and Jacobi iterations for evaluating nonlinear recursions can all be cast under a unifying framework of iterative evaluations of LDSs.
- In Section 3, we discuss how many important problems in machine learning can be framed as evaluating a nonlinear recursion, and we illustrate how the structure of those problems informs which fixed-point methods are best suited for the task.

These contributions unify and clarify several recently proposed methods in the machine learning literature, and they highlight the central importance of linear dynamical systems for parallelizing seemingly sequential processes.

2 Unifying fixed-point iterations using linear dynamical systems

We first introduce notation for evaluating a generic sequence. Let $x_t \in \mathbb{R}^D$ denote the state at time t, and let f_t denote the corresponding transition function at that time point. Throughout this paper, we will use D to denote the dimension of the hidden state and T to denote the sequence length.

Problem Statement (Sequential Evaluation): Evaluate the sequence $\mathbf{x}_{1:T} = (x_1, x_2, \ldots, x_T)$ starting from x_0 via the recursion,

$$x_{t+1} = f_{t+1}(x_t). \tag{1}$$

We omit input dependencies for simplicity, but note that an input u_t can be incorporated into the definition of the transition function by letting $f_{t+1}(x_t) := f(x_t, u_t)$.

The recurrence described in eq. (1) cannot be evaluated in parallel in its original form because x_{t+1} depends directly on x_t, creating a chain of dependencies. As a result, the computation of each state must wait for the previous state to be computed. This approach takes $\mathcal{O}(T)$ time to evaluate the sequence. Moreover, this inherently sequential approach prevents us from fully leveraging modern hardware accelerators, which can dramatically accelerate parallelizable computations. These nonlinear recursions are ubiquitous, appearing, for example, in the denoising pass of a diffusion model, in the forward pass of a nonlinear RNN, or in the recurrence relations in implicit layers and deep equilibrium models.

Fixed-point methods offer a promising alternative: Rather than computing the sequence step by step, make an initial guess for the *entire* trajectory. We denote this initial guess by $\mathbf{x}_{1:T}^{(0)}$. We then iteratively refine this guess, operating over the entire sequence length in parallel, denoting the guess after i fixed-point iterations as $\mathbf{x}_{1:T}^{(i)}$. In particular, the current guess at iteration i is further refined by applying a fixed-point operator $\mathcal{A} : \mathbb{R}^{TD} \mapsto \mathbb{R}^{TD}$ (which depends on the functions f_t and the initial condition x_0) according to

$$\mathbf{x}_{1:T}^{(i+1)} = \mathcal{A}\left(\mathbf{x}_{1:T}^{(i)}\right). \tag{2}$$

In order to be a fixed-point operator, $\mathcal{A}$ should have a unique fixed point $\mathbf{x}_{1:T}^{\star}$, which is the sequential roll-out of eq. (1). Equivalently, $\mathcal{A}$ should satisfy that $\mathbf{x}_{1:T}^{\star} = \mathcal{A}\left(\mathbf{x}_{1:T}^{\star}\right)$, where $\mathbf{x}_{1:T}^{\star}$ is the unique root of the system of equations given by

$$x_{t+1} - f_{t+1}(x_t) = 0 \quad \forall t \in \{0, \ldots, T-1\}. \tag{3}$$

A stopping criterion is used to determine when the fixed point iterations in eq. (2) have converged up to some level of desired numerical accuracy.

Many fixed-point operators can be constructed to satisfy this constraint. However, in the context of parallel evaluation of sequences, we can often be much more specific than a generic operator $\mathcal{A}$. In fact, for the four

Table 1: **Summary of fixed-point iteration schemes as linear dynamical systems**. We order the methods by the number of fixed-point iterations in which they converge (fewer iterations needed to more iterations needed). However, it is important to note that the increased rate of convergence is offset by increased computational and memory requirements of each method. Each iteration is an LDS, i.e. can be written in the form of eq. (4), where A_{t+1} is the transition matrix of the LDS, and an approximation to the Jacobian of the dynamics function $\partial f_{t+1}/\partial x_t$. See Algorithm 1 as well. Throughout we use subscript t to denote sequence position and superscript (i) to indicate fixed-point iteration number. All of these methods are guaranteed to converge in at most T iterations (Gonzalez et al., 2024, Proposition 1). We use "order" to mean the highest number of derivatives taken: Newton and quasi-Newton methods use first derivatives, while Picard and Jacobi methods do not use derivatives of f_t.

Fixed-point method	**Order**	**Transition matrix** A_{t+1}
Newton	first-order	$\dfrac{\partial f_{t+1}}{\partial x_t}\left(x_t^{(i)}\right)$
Quasi-Newton	quasi first-order	$\operatorname{diag}\left[\dfrac{\partial f_{t+1}}{\partial x_t}\left(x_t^{(i)}\right)\right]$
Picard	zeroth-order	I_D
Jacobi	zeroth-order	0

fixed-point methods we consider in this work, the fixed-point operators $\mathcal{A}$ solve a linear time-varying system over the sequence length, with the common form,

$$x_{t+1}^{(i+1)} = f_{t+1}(x_t^{(i)}) + A_{t+1}\,(x_t^{(i+1)} - x_t^{(i)}), \tag{4}$$

where the transition matrix $A_{t+1} \in \mathbb{R}^{D\times D}$ is determined by the dynamics functions f_{t+1} and the current guess for the state $x_t^{(i)}$. Different fixed-point methods use different transition matrices, as shown in Table 1.

The transition matrix A_{t+1} can be thought of as an approximation to the Jacobian of the dynamics function $\partial f_{t+1}/\partial x_t$. Different fixed-point methods simply use different approaches to linearizing the dynamics f_{t+1}. Importantly, because the recursion in eq. (4) is an LDS, it can be evaluated with a parallel scan, which runs in $\mathcal{O}(\log T)$ time on a machine with $\mathcal{O}(T)$ processors (Blelloch, 1990). We provide an introduction to the parallel scan algorithm in Appendix A.

In the rest of this section, we discuss in more detail how the four prominent fixed-point methods discussed in Table 1 reduce to iterative application of LDSs when used to solve a recursion. Fundamentally, all of these methods parallelize nonlinear recursions by iteratively linearizing and evaluating them, as we indicate in Algorithm 1. However, the different methods use different approximations of the Jacobian $\partial f_{t+1}/\partial x_t$ of the dynamics function for their transition matrices A_{t+1}.

Algorithm 1 Fixed-point methods for evaluating sequences using LDSs and parallel scan

procedure ParallelFixedPoint(f, x_0, initial guess $\mathbf{x}_{1:T}^{(0)}$, tolerance ϵ)
  for $i = 0, 1, \ldots, T$ **do**
    $A_{1:T} \leftarrow$ LinearizeDynamics$(f, x_0, \mathbf{x}_{1:T}^{(i)})$ ▷ For all t in parallel
    $\mathbf{x}_{1:T}^{(i+1)} \leftarrow$ EvaluateLDS$(x_0, A_{1:T}, f, \mathbf{x}_{1:T}^{(i)})$ ▷ Using parallel scan
    if ComputeError$(x_0, \mathbf{x}_{1:T}^{(i+1)}, f) < \epsilon$ **then**
      break
  return $\mathbf{x}_{1:T}^{(i+1)}$

2.1 Newton iterations

Danieli et al. (2023), Lim et al. (2024) and Gonzalez et al. (2024) demonstrated that when the fixed-point operator $\mathcal{A}$ is constructed as an appropriately designed LDS, built from a linearization of original nonlinear

recursion f, then each application of $\mathcal{A}$ is equivalent to an iteration of *Newton's root-finding method* on the system of equations given in eq. (3). Specifically, each Newton fixed-point iteration, $\mathbf{x}_{1:T}^{(i+1)} = \mathcal{A}_{\mathrm{N}}(\mathbf{x}_{1:T}^{(i)})$, is defined by the linear recursion,

$$x_{t+1}^{(i+1)} = f_{t+1}(x_t^{(i)}) + \frac{\partial f_{t+1}}{\partial x_t}(x_t^{(i)}) \left(x_t^{(i+1)} - x_t^{(i)}\right), \tag{5}$$

which we recognize as first-order Taylor expansion of the nonlinear recursion. Because this fixed-point iteration uses a first derivative, we refer to it as a *first-order* fixed-point iteration. We note that for Newton iterations, the transition matrix is exactly the Jacobian of the dynamics function, $A_{t+1} = \frac{\partial f_{t+1}}{\partial x_t}(x_t^{(i)})$.

Because eq. (5) is a linear dynamical system, it can be evaluated with a parallel scan. However, each iteration requires $\mathcal{O}(TD^2)$ memory to store the T Jacobian matrices, and $\mathcal{O}(TD^3)$ work to compute the matrix-matrix multiplies in the parallel scan. This expense is prohibitive for large state size or sequence length, which motivates the quasi-Newton iterations we discuss next.

2.2 Quasi-Newton iterations

Because full Newton iterations are costly in compute and memory, there is a wide literature on quasi-Newton methods (Nocedal & Wright, 2006). A particularly simple way to turn the Newton iteration in eq. (5) into a quasi-Newton iteration that uses a parallel associative scan was proposed by Gonzalez et al. (2024): just use the diagonal of the Jacobian of the dynamics function. Specifically, each of these quasi-Newton fixed-point iterations $\mathcal{A}_{\mathrm{QN}}$ is given by

$$x_{t+1}^{(i+1)} = f_{t+1}(x_t^{(i)}) + \mathrm{diag}\left[\frac{\partial f_{t+1}}{\partial x_t}(x_t^{(i)})\right] \left(x_t^{(i+1)} - x_t^{(i)}\right), \tag{6}$$

which we recognize as an LDS with transition matrix $A_{t+1} = \mathrm{diag}\big[\frac{\partial f_{t+1}}{\partial x}(x_t^{(i)})\big]$.

With this transition matrix, the parallel scan requires only $\mathcal{O}(TD)$ space and $\mathcal{O}(TD)$ work, and is therefore more computationally efficient than the full Newton iteration eq. (5). However, quasi-Newton methods take more fixed-point iterations to converge. As discussed by Gonzalez et al. (2024) and Zoltowski et al. (2025) and shown in our experiments in Section 3, whether Newton and quasi-Newton methods run faster for evaluating a given sequence is an empirical question whose answer depends on many factors, including choice of hardware, scale of the problem, and the faithfulness of the approximate Jacobian to the true dynamics. However, the large memory consumption of full Newton iterations renders it infeasible for large-scale problems.

In general, any structured approximation of the Jacobian matrix that remains closed under matrix multiplication[1] could be used to form a parallel quasi-Newton fixed-point iteration for evaluating a sequence. For example, Zoltowski et al. (2025) introduces a parallel quasi-Newton method where each block of the matrix is diagonal. The broader development of quasi-Newton methods that fit into the unifying LDS framework discussed in this paper is an important direction for future work, which we elaborate on in Section 5. However, for simplicity, henceforth in this paper when we refer to "quasi-Newton iterations," we restrict ourselves to the simple diagonal approximation shown in eq. (6).

2.3 Picard iterations

A seemingly different approach to using fixed-point iterations are Picard iterations, which were used by Shih et al. (2023) to parallelize sampling in diffusion models. Picard iterations are often used in the context of evaluating ODEs, where

$$\dot{x} = g(x, t). \tag{7}$$

After Euler discretization with step size Δ, the continuous time eq. (7) becomes the discrete-time recursion,

$$x_{t+1} = x_t + g(x_t, t) \cdot \Delta. \tag{8}$$

[1]Closed in the sense that the product, $A_{t+1}A_t$, has the same structure as A_{t+1} and A_t, as with diagonal matrices. See Appendix A for more detail.

The Picard fixed-point iteration, $\mathbf{x}_{1:T}^{(i+1)} = \mathcal{A}_P(\mathbf{x}_{1:T}^{(i)})$, is then given by,

$$x_{t+1}^{(i+1)} = x_0 + \sum_{s=1}^{t} g(x_s^{(i)}, s) \cdot \Delta. \tag{9}$$

Because Picard iterations do not use any derivatives of the discrete-time recursion, we call them *zeroth-order* fixed-point iterations.

Shih et al. (2023) proves by induction that for any dynamical system given by eq. (8), the fixed-point iterations given by eq. (9) will converge to the true trajectory in at most T iterations. Similarly, Gonzalez et al. (2024) proved that for any dynamical system given by eq. (1), both the Newton fixed-point iterations given by eq. (5) and the quasi-Newton fixed-point iterations given by eq. (6) will converge to the true trajectory in at most T iterations. The similarity of these results and techniques begs the question as to how Picard and Newton iterations relate to each other. Our first result shows that Picard iterations are in fact a type of quasi-Newton iteration, where we approximate the Jacobian of the dynamics function by the identity matrix.

Proposition 1. *The Picard iteration operator $\mathcal{A}_P$ given by eq.* (9) *is a special case of an LDS, eq.* (4)*, where the transition matrix is the identity,*

$$A_{t+1} = I_D.$$

Proof. Define $f_{t+1}(x_t) := x_t + g(x_t, t) \cdot \Delta$. Then, from eq. (9) it follows that

$$\begin{aligned} x_{t+1}^{(i+1)} &= x_t^{(i+1)} + g(x_t^{(i)}, t) \cdot \Delta \\ &= x_t^{(i+1)} - x_t^{(i)} + x_t^{(i)} + g(x_t^{(i)}, t) \cdot \Delta \\ &= f_{t+1}(x_t^{(i)}) + (x_t^{(i+1)} - x_t^{(i)}). \end{aligned}$$

This is exactly of the form of the generic linear recursion shown in eq. (4), with $A_{t+1} = I_D$. □

An important consequence of Proposition 1 is that like Newton iterations and quasi-Newton iterations, Picard iterations can also be cast as an LDS. In Newton iterations, the full Jacobian $\partial f_{t+1}/\partial x_t$ is used in LDS; in quasi-Newton iterations, the diagonal approximation $\mathrm{diag}[\partial f_{t+1}/\partial x_t]$ is used; and in Picard iterations, the identity I_D is used. The Picard iteration is more compute and memory efficient than even quasi-Newton, but is also generally a less faithful approximation and takes more iterations to converge, unless the Jacobian is well-approximated by the identity.

2.4 Jacobi iterations

Yet another seemingly different fixed-point method are Jacobi iterations (Ortega & Rheinboldt, 1970), which were used by Song et al. (2021a) to accelerate computation in a variety of settings in machine learning, such as feedforward networks with skip connections. Jacobi iterations are also a zeroth-order fixed-point method, and are commonly used to solve systems of multivariate nonlinear equations of the form,

$$h_t(\mathbf{x}_{1:T}) = 0 \quad \forall t \in \{1, \ldots, T\}.$$

Instead, the Jacobi fixed-point operator, $\mathbf{x}_{1:T}^{(i+1)} = \mathcal{A}_J(\mathbf{x}_{1:T}^{(i)})$, solves the following system of T *univariate* equations *in parallel* to obtain $\mathbf{x}_{1:T}^{(i+1)}$,

$$h_t^{(i)}\big(x_1^{(i)}, \ldots, x_{t-1}^{(i)}, x_t, x_{t+1}^{(i)}, \ldots, x_T^{(i)}\big) = 0 \quad \forall t \in \{1, \ldots, T\} \tag{10}$$

Song et al. (2021a) considers in particular the problem of solving recurrence relations of the form $x_{t+1} = f_{t+1}(\mathbf{x}_{1:t})$, and proves that, for such a system, Jacobi iterations converge in at most T iterations. This result is directly analogous to similar such results and proofs in Gonzalez et al. (2024) for Newton and quasi-Newton iterations and Shih et al. (2023) for Picard iterations. However, as Song et al. (2021a) notes,

for a *Markovian* system (the setting we consider in this paper), it takes T Jacobi iterations for information from x_0 to propagate to x_T. Thus, Jacobi iterations are not generally suitable for Markovian processes like state space models, RNNs, or sampling from a diffusion model. In fact, in the context of iteratively applying LDSs to parallelize Markovian state space models, we prove that Jacobi iterations are a type of degenerate quasi-Newton iterations, where we "approximate" the Jacobian of the dynamics function by zero.

Proposition 2. *When applied to a Markovian state space model as in eq.* (1)*, the Jacobi iteration operator* $\mathcal{A}_J$ *specified by eq.* (10) *is a special case of the common form, eq.* (4)*, where,*

$$A_{t+1} = 0.$$

Proof. In a Markovian state space model, the recurrence relation always takes the form specified in eq. (1), i.e. $x_{t+1} = f_{t+1}(x_t)$. Thus, Jacobi iterations take the simple form

$$x_{t+1}^{(i+1)} = f_{t+1}(x_t^{(i)}).$$

Because $x_{t+1}^{(i+1)}$ does not depend on $x_t^{(i+1)}$, we see that the transition matrix is zero. □

2.5 Summary

We have shown how important parallel fixed-point iterations—Newton, quasi-Newton, Picard, and Jacobi iterations—can all be cast as LDSs when deployed for evaluating nonlinear recursions, as summarized in Table 1. The regimes where these different methods excel are therefore dictated by the form of the Jacobians of their dynamics functions: if each f_{t+1} is close to an identity update (as is the case in sampling from a diffusion model with small discretization parameter), then Picard will excel; if the dynamics are nearly uncoupled across state dimensions, then quasi-Newton using a diagonal approximation will excel; and if the dynamics have multiple dependencies across coordinates and the dimension D is not too large, then Newton iterations will excel. On a Markovian state space model, Jacobi iterations require T iterations for information to propagate from the initial condition to the end of the sequence, the same as sequential evaluation. However, we include Jacobi iterations for completeness, noting that they can be useful in non-Markovian systems, such as architectures with skip connections.

An important corollary is that because all of these fixed-point iterations can be cast as LDSs, they are all guaranteed to converge in all problem settings in at most T iterations (Gonzalez et al., 2024, Proposition 1). However, as we noted above, the precise convergence rates of the different fixed-point methods will be problem dependent. In the next section, we develop these observations further by discussing how a wide variety of important tasks in machine learning can be understood as the evaluation of Markovian state space models. Furthermore, we show how the unifying framework proposed in this section helps us understand which fixed-point iterations will excel in which problem settings.

3 Case studies on when to use each kind of fixed-point iteration

Many important tasks in machine learning can be understood as the evaluation of a sequential process. For example, evaluating deep neural networks, recurrent neural networks, state space models, or transformers (sequential evaluation of the blocks over depth); sampling from a diffusion model, or in general any Markov chain or process such as Markov chain Monte Carlo; maintaining a world model over the course of many sequential updates to the state; and many more problems fall under this heading. In this section, we consider three empirical case studies that illustrate how the unifying framework above provides heuristic guidance about which fixed-point schemes will excel in which settings. This concordance is based on the structure of the Jacobian of f_{t+1} and the relative computational cost of different fixed-point methods.

Our heuristic guidance, in a nutshell, is:

> *Use the simplest approximate Jacobian as possible, but no simpler.*

Simpler approximate Jacobians, like the identity matrix Picard iterations, are less computationally expensive, meaning that each fixed-point iteration is more efficient. So, if the lower-order fixed-point method still converges in a small number of fixed-point iterations, it achieves the sequential roll-out $\mathbf{x}^*$ in faster wall-clock times on GPUs than higher-order fixed point methods. However, if the higher-order fixed-point method (e.g. Newton or quasi-Newton) converges in far fewer iterations than the lower-order fixed-point method, then the increased computation of the higher-order fixed-point method is worthwhile. Intuitively, the number of iterations needed for a fixed-point method to converge should be related to how faithfully A_t approximate the linearized dynamics $\partial f_{t+1}/\partial x_t$. We support this intuition with three empirical case studies to highlight where Newton, quasi-Newton, and Picard iterations excel. Further experimental details are provided in Appendix B.

3.1 Case study #1: Solving the group word problem with Newton iterations

Newton iterations should outperform quasi-Newton and Picard iterations in settings where the Jacobian of the recursion, f_{t+1}, is not well approximated by its diagonal or by the identity matrix. One example of such a recursion is the *group word problem*, which has been used to theoretically and empirically assess the limits of sequential modeling architectures for state-tracking tasks (Kim & Schuster, 2023; Liu et al., 2023; Merrill et al., 2024; Grazzi et al., 2025; Schöne et al., 2025). In the sequence-modeling community, the term "group word problem" is defined as follows.

Definition 1 (Group Word Problem)**.** *Let G be a finite group and let $g_1, g_2, \ldots, g_T$ be a sequence of group elements. The group word problem is to evaluate the product $g_1 \cdot g_2 \cdots g_T$. Since each $g_t \in G$, the product of these group elements belongs to G as well.*

Merrill et al. (2024) emphasizes that nonlinear RNNs in both theory and practice are able to learn the group word problem in arbitrary groups to high accuracy with only a single layer, whereas compositions of popular linear RNNs linked by nonlinearities, such as S4 (Gu et al., 2022) and Mamba[2] (Gu & Dao, 2024), require a number of layers that grows with T. Merrill et al. (2024) emphasizes that recurrent architectures with nonlinear transitions are well-suited for solving the group word problem, because in theory and practice, such architectures can learn the group word problem to high accuracy with a single layer. Other literature has explored the value of matrix-valued states (Beck et al., 2024; Grazzi et al., 2025). However, in Proposition 3 below, we show that neither nonlinearity nor matrix-valued states are needed to understand or solve the group word problem. Instead, the problem can be formulated as an LDS with vector-valued states and input-dependent transition matrices.

Proposition 3. *Let G be a finite group. Then there exists some $D \leq |G|$ for which we can represent the group word problem as a time-varying LDS, $f_{t+1}(x_t) = A_{t+1}x_t$, with states $x_t \in \mathbb{R}^D$ denoting the running product of group elements and transition matrices $A_{t+1} \in \mathbb{R}^{D\times D}$ that depend on the input g_{t+1}.*

Proof. By Cayley's theorem, any finite group G can be embedded in a symmetric group S_D, for some $D \leq |G|$. Therefore, by choosing the initial state $x_0 \in \mathbb{R}^D$ to have D distinct entries (a "vocabulary" of size D), we can use the tabular representation of permutations (Artin, 2011, eq. 1.5.2) to represent an element of S_D as x_t (by a permutation of the elements of x_0). We can also choose $A_{t+1} \in \mathbb{R}^{D\times D}$ to be the permutation matrix corresponding to the embedding of g_{t+1} in S_D, since any element of S_D can be represented as a $D \times D$ permutation matrix (e.g., see Figure 1B). Consequently $x_t = A_tA_{t-1}\ldots A_2A_1x_0$ is an embedding of an element of G in S_D in the tabular representation. In fact, $x_t \in \mathbb{R}^D$ represents the running product $g_1g_2\ldots g_{t-1}g_t$, which is precisely the goal of the group word problem. □

Though we have cast the group word problem as a time-varying LDS with $f_{t+1}(x_t) = A_{t+1}x_t$, note that we can still evaluate this recursion with any of the fixed-point methods described above. Since the dynamics are linear, the Newton iteration corresponds to evaluating the LDS with a parallel scan, and it converges in one iteration. While quasi-Newton or Picard methods would require more iterations to converge, they could still be more efficient in wall-clock time, since they use less memory and compute per iteration. However, since the

[2]Mamba allows input-dependent dynamics matrices but they must be diagonal, which prevents a single Mamba layer from implementing the particular LDS in Proposition 3, which uses permutation matrices.

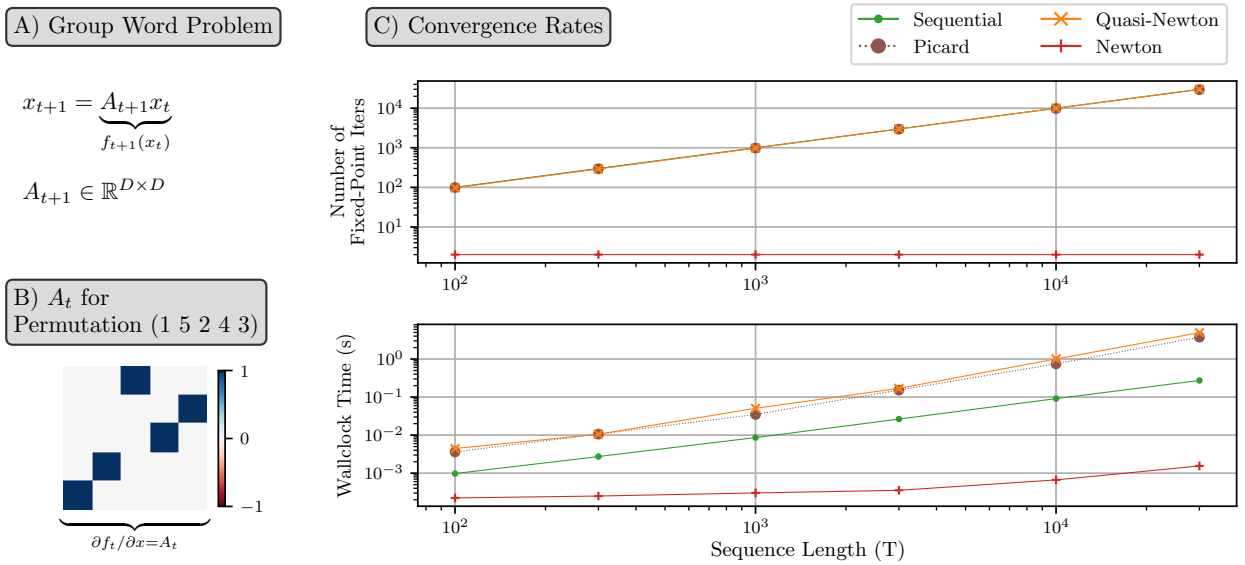

Figure 1: **A single Newton iteration solves the S_5 group word problem, whereas the number of iterations required for the quasi-Newton and Picard methods increases with problem size.** We consider the task of evaluating the product of S_5 group elements. **A:** The group word problem can be expressed as an LDS with input-dependent state-transition matrices. **B:** An example input-dependent transition matrix A_t for permutation (1 5 2 4 3), in cycle notation. **C:** For each fixed-point method and a range of sequence lengths, T, we compute the median (over ten random seeds) number of fixed-point iterations to converge (top) and the median wall-clock time (bottom). While a single Newton iteration is sufficient to solve the S_5 problem, the number of iterations required for the Picard and quasi-Newton methods increases with the sequence length.

input-dependent state transition matrices of the group word problem have mostly non-diagonal entries, as they are permutation matrices, we expect the diagonal and identity transition matrices in the quasi-Newton and Picard iterations, respectively, to be poor approximations of the true dynamics, and therefore these methods should take a large number of fixed-point iterations to converge.

We support this hypothesis with a simple experiment simulating the S_5 word problem, a standard problem in the sequence modeling literature (Merrill et al., 2024; Grazzi et al., 2025). In this setting, Figure 1 shows that quasi-Newton and Picard iterations require nearly T iterations to converge. On the other hand, we see that Newton's method solves the S_5 word problem with just one fixed-point iteration, as expected since the true dynamics are linear. The speed-up is also apparent in the wall-clock time comparison, where we see that Newton is faster than quasi-Newton, Picard, and sequential evaluation, regardless of T.

3.2 Case Study #2: Parallelizing RNNs with quasi-Newton iterations

We next consider a task where first-order fixed-point iterations like quasi-Newton and Newton iterations do well, but zeroth-order methods like Picard do not. This task is parallelizing recurrent neural networks (RNNs), like the Gated Recurrent Unit or GRU (Cho et al., 2014).

We show the results of a simple experiment in Figure 2. We evaluate GRUs with random parameter initialization for different hidden dimension sizes D and sequence lengths T using sequential evaluation as well as zeroth-order (Picard) and first-order (Newton and quasi-Newton) iterations. Because the GRU in general has complicated dynamics that are not well-approximated by the identity, we observe that Picard iterations take a prohibitively large number of fixed-point iterations to converge (effectively equal to the sequence length). Therefore, Picard iterations are not suitable for these classes of dynamical systems. By contrast, first-order fixed-point iterations can yield orders of magnitude speed-ups over sequential evaluation.

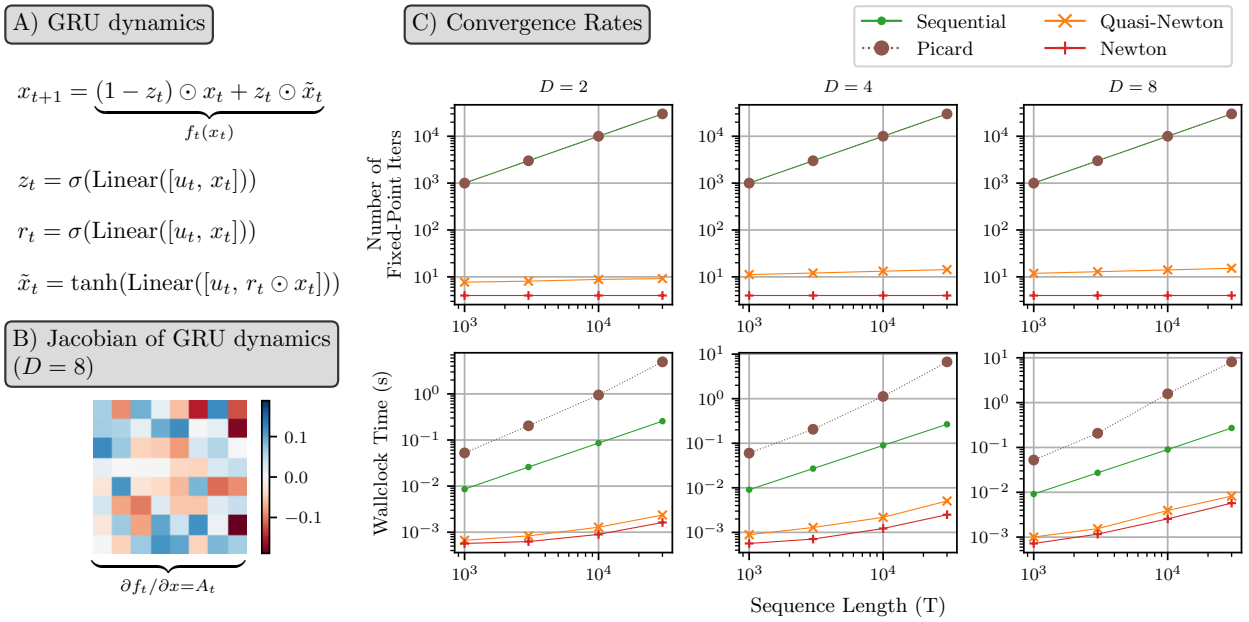

Figure 2: **First-order iterations can excel for complicated dynamics.** We evaluate GRUs with random parameter initialization for different sequence lengths T and hidden state sizes D. **A:** The nonlinear dynamics of a GRU, following Feng et al. (2024), where x_t is the hidden state, u_t is the input, and the notation Linear$[\cdot, \cdot]$ indicates a linear readout from the concatenation of two vectors. **B:** A representative Jacobian matrix $\partial f_t / \partial x$ from a GRU trajectory, which is not well approximated by the identity matrix. **C:** For each fixed-point method and a range of sequence lengths, T, and state sizes, D, we compute the median (over ten random seeds) number of fixed-point iterations to converge (top row) and the median wall-clock time (bottom row). While first-order methods yield order-of-magnitude speed-ups over sequential evaluation, Picard iterations take nearly T iterations to converge.

3.3 Case Study #3: Parallelizing discretized Langevin diffusion with Picard iterations

Based on Table 1, we expect that if the Jacobian of the dynamics function is well-approximated by the identity matrix, then Picard should converge relatively quickly and at considerably lower cost. A canonical example of such a system comes from Langevin dynamics (Langevin, 1908; Friedman, 2022), which are a workhorse for MCMC (Besag, 1994) and motivated the development of score-matching methods (Song & Ermon, 2019), which are closely related to diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021b).

In Langevin dynamics for a potential ϕ, the state x_t evolves according to a nonlinear recursion with dynamics,

$$f_{t+1}(x_t) = x_t - \epsilon \nabla \phi(x_t) + \sqrt{2\epsilon} w_t,$$

where ϵ is the step size and $w_t \overset{\text{iid}}{\sim} \mathcal{N}(0, I_D)$. Since the Jacobian is, $\frac{\partial f_{t+1}}{\partial x_t}(x_t) = I_D - \epsilon \nabla^2 \phi(x_t)$, it follows that the identity approximation should work well with small step-sizes, ϵ. More generally, the identity approximation tends to be well-suited to problems where a differential equation is discretized with small step sizes, such as when as sampling from diffusion models (Holderrieth & Erives, 2025). In these settings, the heuristic guidance from Table 1 suggests that Picard iterations will enjoy reasonably fast convergence.

We illustrate this heuristic guidance with a simple experiment shown in Figure 3. We simulate Langevin dynamics on a potential ϕ given by the negative log probability of the mixture of two anisotropic Gaussians. In this setting, Picard iterations take far fewer than T iterations to converge and can be faster than sequential evaluation. We note that quasi-Newton iterations, which include information only about the diagonal of the

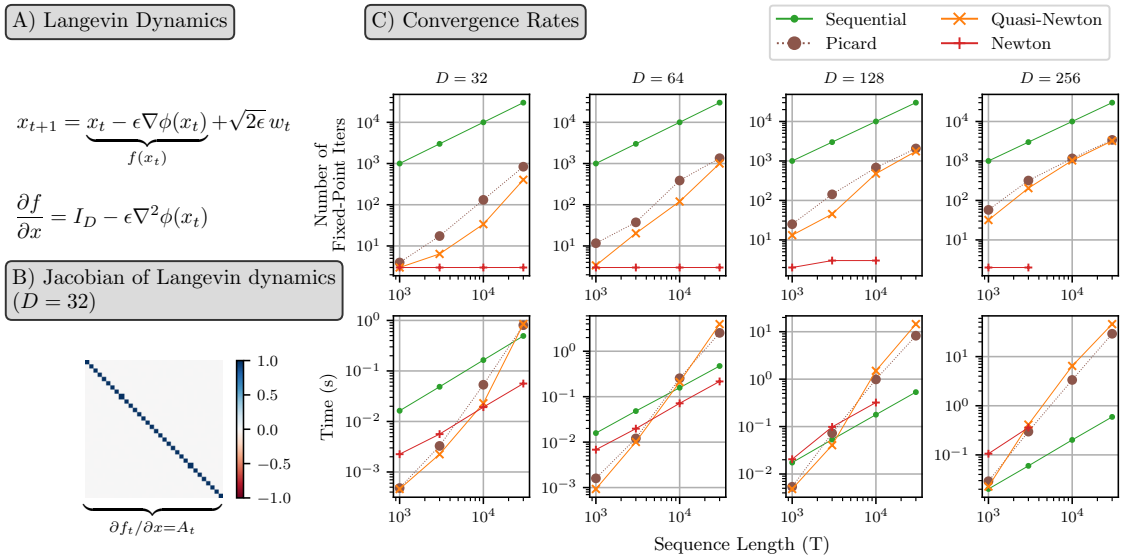

Figure 3: **Zeroth-order fixed-point iterations work well when the Jacobian is close to the identity.** We evaluate Langevin dynamics for a potential ϕ. **A:** The nonlinear dynamics of Langevin dynamics for a potential ϕ and step size ϵ, where x_t is the state and w_t is Gaussian noise. **B:** The Jacobian for Langevin dynamics is well-approximated by the identity matrix, especially for small step size $\epsilon = 1 \times 10^{-5}$. **C:** We evaluate Langevin dynamics for larger dimensions, plotting the median of 10 random seeds. Picard iterations converge in a relatively small number of fixed-point iterations, as the Jacobian is well-approximated by an identity matrix. Here, zeroth- and first-order methods are often comparable in wall-clock time. The missing Newton iteration points indicate the GPU ran out of memory.

Jacobian of the dynamics, appear to have comparable wall-clock time, by virtue of taking fewer iterations to converge (though each fixed-point iteration involves more work).

Whether fixed-point iterations are faster than sequential evaluation also depends on memory utilization. For example, Shih et al. (2023) demonstrated wall-clock speed-ups when using Picard iterations for sampling from a diffusion model using a "sliding window" to only evaluate chunks of the sequence length where the parallel scan algorithm can fit in memory. We discuss these considerations further in Appendix B.4. However, this simple experiment simply aims to show that there are settings where zeroth-order fixed-point iterations can outperform sequential evaluation or first-order fixed-point iterations. We leave more hardware-aware implementations for future work.

4 Related Work

In this paper we unify prominent fixed-point methods for the parallel evaluation of sequences in the language of linear dynamical systems. While many papers have employed different fixed-point iterations for different problems in machine learning — Lim et al. (2024) and Danieli et al. (2023) using Newton iterations, Gonzalez et al. (2024) using quasi-Newton iterations, Shih et al. (2023) using Picard iterations, and Song et al. (2021a) using Jacobi iterations, among other works — to the best of our knowledge no one has explicitly unified these different methods in the language of linear dynamical systems.

General unification of fixed-point methods: parallel-chord methods While connections between Newton's method and Picard iterations have been made before outside of the machine learning literature, our contribution is the tight coupling of these methods to LDSs in the context of parallel evaluation of nonlinear sequences. Ortega & Rheinboldt (1970, Ch. 7) considered the problem of solving a nonlinear equation $F(\mathbf{x}) = 0$. They showed that Newton and Picard iterations are special cases of what they call

parallel-chord methods[3], where each iterate is given by

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - T^{-1}F(\mathbf{x}^{(i)}),$$

for some matrix T, and they proved convergence rates for these methods. We discuss the relationship between the unifying frameworks put forward in Ortega & Rheinboldt (1970) and in our paper at greater length in Appendix C.1. The primary difference is that by focusing on the setting of nonlinear sequence evaluation, we bring into greater focus the role of the Jacobian of the dynamics function. Moreover, by unifying fixed-point iterations in the language of LDSs, we emphasize their parallelizability over the sequence length using the parallel scan (Blelloch, 1990).

Other fixed-point methods: mixing sequential with parallel In this note, we focus on Jacobi, Picard, and Newton iterations because of their prominence (Song et al., 2021a; Shih et al., 2023; Danieli et al., 2023; Lim et al., 2024; Gonzalez et al., 2024; Grazzi & Zanella, 2025; Zoltowski et al., 2025) and their relationship to LDSs, as listed in Table 1. However, there is a wide literature on iterative solvers (Ortega & Rheinboldt, 1970; Young, 2014). Many of these other methods can also be parallelized over the sequence length, or provide a mixture of parallel and sequential computation. For example, Song et al. (2021a) considers Gauss-Seidel iterations. Although Gauss-Seidel iterations reduce to sequential evaluation when applied to Markovian processes, Song et al. (2021a) also emphasize how the structure of the problem and hardware considerations dictate the optimal mixture of parallel and sequential computation. Parareal iterations mix parallel and sequential computation by applying parallelization at multiple length scales, and have also been used to parallelize diffusion models (Selvam et al., 2024). Tang et al. (2024) also parallelized diffusion models using both a generalization of Jacobi iterations, as well as Anderson acceleration (Anderson, 1965; Walker & Ni, 2011), which they modify to be a form of quasi-Newton. We discuss other fixed-point methods in further detail in Appendix C.2.

5 Discussion

This work unified a variety of approaches for parallelizing recursions via fixed-point iterations — including zeroth-order methods like Jacobi and Picard iterations as well as first-order methods like Newton and quasi-Newton iterations — under a common framework. In each case, the iterates reduce to evaluating an appropriately constructed linear dynamical system, which approximates the nonlinear recursion of interest. Moreover, we have demonstrated how this unifying framework provides insight into which different problems in machine learning are likely to benefit from which types of fixed-point iterations.

Clarifying the relationships and properties of these approaches through the lens of linear dynamical systems also suggests promising areas for future study. One clear direction of future work is to explore additional approaches for exploiting problem-specific structure. For example, an intermediate between Picard and quasi-Newton methods is a scaled identity approximation, $A_{t+1} = a_{t+1}I_D$. If we had prior knowledge on the appropriate scaling factors, $a_{t+1} \in \mathbb{R}$, we could avoid computing any Jacobian-vector product evaluations. More generally, there exist other groups of structured matrices with compact representations that are closed under composition such that a parallel evaluation of the LDS would be computationally efficient. Examples include permutation matrices, block-diagonal matrices, and block matrices where each sub-block is diagonal, among others. Future work should enumerate these use cases and investigate problem-specific applications where they are appropriate. One example application is for more efficient parallelization of the group word problem using a compact representation of permutation matrices.

In conclusion, understanding the shared backbone of these fixed-point methods can also give practitioners guidance about which methods to use for which problems. As parallel evaluation of seemingly sequential processes becomes increasingly important in machine learning, these insights may provide valuable guidance to the field.

[3]The term "parallel" here does not have to do with parallelized computation over a sequence length, but rather comes from a geometric perspective on Newton's method: solve a nonlinear equation by iteratively solving a linear approximation to it, i.e. forming a "chord" that is "parallel" to the nonlinear function at our current guess for its root.

References

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12 (4):547–560, 1965.

Michael Artin. *Abstract Algebra*. Pearson, 2nd edition, 2011. ISBN 9780132413770.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended Long Short-Term Memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Julian Besag. Comment on "Representations of knowledge in complex systems" by Grenander and Miller. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(4):549–581, 1994.

Guy E. Blelloch. Prefix Sums and Their Applications. Technical Report CMU-CS-90-190, Carnegie Mellon University, School of Computer Science, 1990.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Federico Danieli, Miguel Sarabia, Xavier Suau, Pau Rodríguez, and Luca Zappella. DeepPCR: Parallelizing Sequential Operations in Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Mónika Farsang, Ramin Hasani, Daniela Rus, and Radu Grosu. Scaling up liquid-resistance liquid-capacitance networks for efficient sequence modeling. *arXiv preprint arXiv:2505.21717*, 2025.

Kayvon Fatahalian and Kunle Olukotun. Lecture 8: Data-parallel thinking. Lecture slides, CS149: Parallel Computing, Stanford University, 2024.

Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadeghi. Were RNNs All We Needed? *arXiv*, 2024.

Roy Friedman. A Simplified Overview of Langevin Dynamics. Blog post, 2022.

V. A. Gasilov, V. F. Tishkin, A. P. Favorskii, and M. Yu. Shashkov. The use of the parallel-chord method to solve hydrodynamic difference equations. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 21(3):178–192, 1981. ISSN 0041-5553. doi: 10.1016/0041-5553(81)90075-6.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. *arXiv preprint arXiv:2502.05171*, 2025.

Xavier Gonzalez, Andrew Warrington, Jimmy T. H. Smith, and Scott W. Linderman. Towards Scalable and Stable Parallelization of Nonlinear RNNs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Xavier Gonzalez, Leo Kozachkov, Kenneth L. Clarkson, David M. Zoltowski, and Scott W. Linderman. Predictability Enables Parallelization of Nonlinear State Space Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. MIT Press, 2016.

Riccardo Grazzi, Julien Siems, Jörg KH Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues. In *International Conference on Learning Representations (ICLR)*, 2025.

Sebastiano Grazzi and Giacomo Zanella. Parallel computations for Metropolis Markov chains with Picard maps, 2025.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Conference on Language Modeling (COLM)*, 2024.

Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *The International Conference on Learning Representations (ICLR)*, 2022.

Mark Harris, Shubhabrata Sengupta, and John D. Owens. Parallel prefix sum (scan) with CUDA. In Hubert Nguyen (ed.), *GPU Gems 3*, chapter 39, pp. 851–876. Addison-Wesley Professional, Upper Saddle River, NJ, August 2007.

Syeda Sakira Hassan, Simo Särkkä, and Ángel F García-Fernández. Temporal parallelization of inference in hidden markov models. *IEEE Transactions on Signal Processing*, 69:4875–4887, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Franz A. Heinsen. Efficient parallelization of a ubiquitous sequential computation, 2023.

W Daniel Hillis and Guy L Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12): 1170–1183, 1986.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Peter Holderrieth and Ezra Erives. Introduction to flow matching and diffusion models, 2025. MIT course.

Amber Hu, Henry Smith, and Scott Linderman. SING: SDE Inference via Natural Gradients. *arXiv preprint arXiv:2506.17796*, 2025.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *CoRR*, abs/2111.00254, 2021.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. *arXiv preprint arXiv:2305.02363*, 2023.

Volodymyr Kyrylov. Accelerated scan, 2024. URL `https://github.com/proger/accelerated-scan`. GitHub repository.

Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4): 831–838, 1980.

Sivaramakrishnan Lakshmivarahan and Sudarshan K Dhall. *Parallel computing using the prefix problem*. Oxford University Press, 1994.

Paul Langevin. On the Theory of Brownian Motion. *American Journal of Physics*, 65(11):1079–1081, 1908. English translation, introduced by D. S. Lemons and translated by A. Gythiel. Original: C. R. Acad. Sci. 146, 530–533 (1908).

Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. In *International Conference on Learning Representations (ICLR)*, 2024.

Scott W Linderman, Peter Chang, Giles Harper-Donnelly, Aleyna Kara, Xinglong Li, Gerardo Duran-Martin, and Kevin Murphy. Dynamax: A Python package for probabilistic state space modeling with JAX. *Journal of Open Source Software*, 10(108):7069, 2025.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

Eric Martin and Chris Cundy. Parallelizing Linear Recurrent Neural Nets Over Sequence Length. In *International Conference on Learning Representations (ICLR)*, 2018.

William Merrill, Jackson Petty, and Ashish Sabharwal. The Illusion of State in State-Space Models. In *International Conference on Machine Learning (ICML)*, 2024.

Kevin P Murphy. *Probabilistic machine learning: Advanced topics*. MIT Press, 2023.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2 edition, 2006.

James M Ortega and Werner C Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York and London, 1970. Republished by SIAM in 2000.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting Recurrent Neural Networks for Long Sequences. In *International Conference on Machine Learning (ICML)*, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965.

Simo Särkkä and Ángel F. García-Fernández. Temporal parallelization of bayesian smoothers. *IEEE Transactions on Automatic Control*, 66(1):299–306, 2021. doi: 10.1109/TAC.2020.2976316.

Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*, volume 17. Cambridge University Press, 2023.

Felix Sarnthein. Linear recurrences accessible to everyone. In *ICLR Blogposts*, 2025.

Mark Schöne, Babak Rahmani, Heiner Kremer, Fabian Falck, Hitesh Ballani, and Jannes Gladrow. Implicit Language Models are RNNs: Balancing Parallelization and Expressivity. In *International Conference on Machine Learning (ICML)*, 2025.

Nikil Roashan Selvam, Amil Merchant, and Stefano Ermon. Self-Refining Diffusion Samplers: Enabling Parallelization via Parareal Iterations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel Sampling of Diffusion Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jimmy T.H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015.

Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating Feedforward Computation via Parallel Nonlinear Equation Solving. In *International Conference on Machine Learning (ICML)*, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations (ICLR)*, 2021b.

Harold S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM*, 20(1):27–38, 1973.

Simo Särkkä and Lennart Svensson. Levenberg-Marquardt and line-search extended Kalman smoothers. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5875–5879. IEEE, 2020.

Zhiwei Tang, Jiasheng Tang, Hao Luo, Fan Wang, and Tsung-Hui Chang. Accelerating Parallel Sampling of Diffusion Models. In *International Conference on Machine Learning (ICML)*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

Fatemeh Yaghoobi, Adrien Corenflos, Sakira Hassan, and Simo Särkkä. Parallel square-root statistical linear regression for inference in nonlinear state space models. *SIAM Journal on Scientific Computing*, 47(2): B454–B476, 2025.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Attention Transformers with Hardware-Efficient Training. In *International Conference on Machine Learning (ICML)*, 2024.

David M. Young. *Iterative Solution of Large Linear Systems*. Elsevier, 2014. ISBN 978-0-12-773050-9.

Yixiu Zhao and Scott Linderman. Revisiting structured variational autoencoders. In *International Conference on Machine Learning (ICML)*, 2023.

David M. Zoltowski, Skyler Wu, Xavier Gonzalez, Leo Kozachkov, and Scott W. Linderman. Parallelizing MCMC Across the Sequence Length. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

A The Parallel Scan: A Gentle Introduction

A.1 A very simple example: multiplying matrices

The *parallel scan* (Stone, 1973; Blelloch, 1990), also known as the *associative* scan and, colloquially, *pscan*, is a well-known primitive in the parallel computing literature (Hillis & Steele Jr, 1986; Ladner & Fischer, 1980; Lakshmivarahan & Dhall, 1994). The core idea of the parallel scan is a divide-and-conquer algorithm. We illustrate this point in the simple example of multiplying a series of matrices together.

Simple Example (Multiplying Matrices): Given a series of square matrices $A_1, A_2, \ldots, A_{T-1}, A_T$, compute their product[4], $A_T A_{T-1} \ldots A_2 A_1$. The simplest way to carry out the matrix multiplication is sequentially: first compute A_1, then compute $A_2 A_1$, then compute $A_3 A_2 A_1$, and so on. Such an approach takes $\mathcal{O}(T)$ time.

A core insight of the parallel scan is that matrix multiplication is *closed*; that is, if $A_s \in \mathbb{R}^{D \times D}$ and $A_t \in \mathbb{R}^{D \times D}$, then $A_t A_s \in \mathbb{R}^{D \times D}$. Thus, matrix products can be computed recursively in pairs, as illustrated in Figure 4.

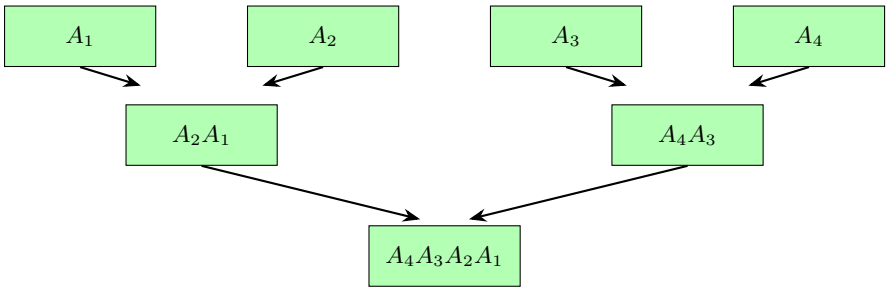

Figure 4: **Parallel Scan for Matrix Multiplication.** We illustrate a divide-and-conquer approach to compute the product $A_4 A_3 A_2 A_1$. Note that this divide-and-conquer approach naturally leads to $\mathcal{O}(\log T)$ depth.

Because of the divide-and-conquer (binary-tree-like) nature of this approach to multiplying matrices, with $\mathcal{O}(T)$ processors, the time needed to get the matrix product is only $\mathcal{O}(\log T)$. This simple example illustrates the core intuition behind the parallel scan: a closed operation leading to a divide-and-conquer approach that parallelizes a computation so that it takes sublinear time. However, there are two additional details of the parallel associative scan that we should address: arbitrary binary associative operators and closure; and getting intermediate products.

A.2 Detail #1: Parallel scans for arbitrary binary associative operators

Matrix multiplication is an associative operator, as $A_3(A_2 A_1) = (A_3 A_2)A_1$. In general, consider a binary associative operator $\otimes$, which would satisfy $q_3 \otimes (q_2 \otimes q_1) = (q_3 \otimes q_2) \otimes q_1$. Now, let us further assume that this binary associative operator closed:

Definition 2 (Closure)**.** *A binary associative operator $\otimes$ is closed over a set $\mathcal{S}$ if it satisfies the property:*

$$q_1 \in \mathcal{S}, q_2 \in \mathcal{S} \Rightarrow q_2 \otimes q_1 \in \mathcal{S}. \tag{11}$$

If $\otimes$ is closed, then we can again use a parallel scan to compute the cumulative product of the operands. A wide range of binary associative operators are closed, and can thus be parallelized with the parallel scan. Some examples include:

[4]Note that we have the matrices act via left-multiplication over the sequence length, because this is the most common way to write matrix-vector products.

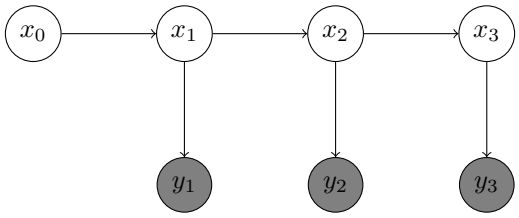

Figure 5: **A linear Gaussian state space model (LGSSM):** The LGSSM consists of latent variables x_t and observed variables y_t. The generative model of the LGSSM consists of dynamics $x_{t+1} \sim \mathcal{N}(Ax_t, Q)$ and emissions $y_{t+1} \sim \mathcal{N}(Hx_{t+1}, R)$.

- **Scalar addition:** The fact that addition of scalars (and vectors) is closed allows cumulative sums to be computed with the parallel scan algorithm. In this instance, it is also known as the *prefix sum* algorithm. Clearly, addition is associative and closed, and so summing a series of scalars can be done with a divide-and-conquer approach.

- **Composition of affine functions, as in LDSs.** Consider the affine function $f_i(x) = A_i x + b_i$. Notice that the composition of affine functions is also affine, as $f_j(f_i(x)) = A_j A_i x + (b_j + A_j b_i)$. Thus, if we represent the operands as ordered pairs (A_i, b_i) and (A_j, b_j), we can write the associative operator $\otimes$ for the composition of affine functions as

  $$(A_i, b_i) \otimes (A_j, b_j) = (A_j A_i, b_j + A_j b_i).$$

  Thus, we observe that in this setting, $\otimes$ is closed. We also should check that $\otimes$ is associative: we can do so with either elementary algebra, or by observing that function composition is associative.

  This observation that composition of affine functions can be parallelized with the associative scan is what lets us parallelize LDSs. The parallelization of LDSs is what allows for parallel computation in many important architectures in sequence modeling, such as linear RNNs (Martin & Cundy, 2018; Orvieto et al., 2023), deep SSMs (Smith et al., 2023; Gu & Dao, 2024), and nonlinear[5] RNNs (Lim et al., 2024; Gonzalez et al., 2024; Farsang et al., 2025). This parallel scan for LDSs is the core primitive uniting the fixed-point methods discussed in this paper.

- **Kalman filtering and smoothing:** Parallel scans can also be utilized in probabilistic modeling. A standard probabilistic model is the *linear Gaussian state space model* (LGSSM), where the latent variables x_t follow linear dynamics with Gaussian noise, and emit observations y_t with linear readouts with Gaussian noise (Murphy, 2023; Särkkä & Svensson, 2023). See Figure 5.

  Two canonical inferential targets in the LGSSM are the filtering distributions, $p(x_t \mid y_{1:t})$, and the smoothing distributions, $p(x_t \mid y_{1:T})$. The Kalman filter (Kalman, 1960) and Rauch-Tung-Striebel (RTS) smoother (Rauch et al., 1965) obtain the filtering and smoothing distributions (respectively) in an LGSSM. The Kalman filter makes a single pass forward in time to get the filtering distributions, while the RTS smoother then makes an additional pass backwards in time to get the smoothing distributions. Both the Kalman filter and RTS smoother would seem to be inherently sequential algorithms, requiring $\mathcal{O}(T)$ time. However, Särkkä & García-Fernández (2021) demonstrated that the Kalman filter and RTS smoother can also be parallelized over the sequence length via the construction of custom binary associative operators and a parallel scan. While we leave the details of this construction to Särkkä & García-Fernández (2021), we note that it is intuitively plausible to be able to parallelize filtering and smoothing in an LGSSM with a parallel scan because

  – the dynamical backbone is an LDS, for which we have a parallel scan;
  – since everything is linear and Gaussian, all distributions remain Gaussian, hinting at closure; and

[5]The parallel scan is used in nonlinear RNNs via the iterative fixed-point methods discussed in this paper, i.e. Newton and quasi-Newton iterations, see Algorithm 1.

– we can combine $p(x_s|x_0, y_{1:s})$ with $p(x_t|x_s, y_{s+1:t})$ to obtain $p(x_t|x_0, y_{1:t})$, suggesting a divide-and-conquer strategy.

These parallel filtering and smoothing algorithms are useful in machine learning, allowing for parallelization of structured variational autoencoders (Johnson et al., 2016; Zhao & Linderman, 2023). Similar approaches also work for Hidden Markov Models (Hassan et al., 2021) and for computing log-normalizing constants (Hu et al., 2025).

- **Parallelizing fixed point iterations:** Finally, these parallel filtering and smoothing are directly applicable to the types of parallel fixed-point iterations that are the focus of this paper. In particular, the ELK algorithm (Gonzalez et al., 2024)—**E**valuating **L**evenberg-Marquardt via **K**alman—stabilizes the Newton iterations (Lim et al., 2024) we discuss in this paper using the Levenberg-Marquardt (trust-region) method (Levenberg, 1944; Marquardt, 1963). The trust-region updates are able to be computed using a parallel scan because they are equivalent to a Kalman smoother in an appropriately constructed LGSSM (Särkkä & Svensson, 2020).

What about arbitrary function composition? The astute reader might note that the composition of functions, i.e. $f_1 \circ f_2$, is *always* a binary associative operator. So, why do we have all these special cases of parallel scans, and not simply one parallel scan for the composition $\circ$ of arbitrary functions f_i?

The reason to have many different parallel scans is precisely the importance of having the binary associative operator be *closed*. In all the previous examples, the binary associative operator $\otimes$ satisfies Definition 2, letting us easily store combinations of operands $q_i \otimes q_j$ and so employ a divide-and-conquer technique.

While we could consider some gigantic function space $\mathcal{F}$, for which function composition would be closed, the practical question then becomes: how would we store the combinations of operands? If we do not have some compact representation for elements of $\mathcal{F}$, then we cannot use a parallel scan in practice, even though the parallel scan may seem applicable in theory.

A.3 Detail #2: Obtaining the intermediate terms in the product

When we evaluate a linear dynamical system, we often do not want only the final term x_T, but also all the intermediate terms $\mathbf{x}_{1:T}$, i.e. the full trajectory. So far, the parallel scan as presented would only yield the final term x_T as well as intermediate terms that happened to be powers of 2, i.e. x_1, x_2, x_4, x_8, etc.

However, the parallel scan can be easily adjusted to obtain all the intermediate terms as well. Let us return to our very simple example of matrix multiplication to illustrate, in particular the setting where $T = 8$. We will denote the individual matrices as $A_1, A_2, A_3, \ldots A_8$, and their products as $A_{s:t}$, i.e. $A_{5:6} = A_6 A_5$.

The first phase of the parallel scan is the *up-sweep*, and takes $\log(T)$ iterations and $\mathcal{O}(T)$ memory. We start multiplying adjacent pairs of matrices together, going, for example[6], from A_8 to $A_{7:8}$ to $A_{5:8}$ to $A_{1:8}$.

Then, in the *down-sweep*, we fill in the missing products to obtain all the cumulative products $A_{1:t}$ for $1 \leq t \leq T$. Intuitively, the down-sweep also takes $\mathcal{O}(\log T)$ iterations, for the same reason that any natural number T can be represented using $1 + \log_2(T)$ digits in binary.

Table 2: **Up-sweep** for multiplying $A_1, A_2, \ldots A_8$.

Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8
A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
A_1	$A_{1:2}$	A_3	$A_{3:4}$	A_5	$A_{5:6}$	A_7	$A_{7:8}$
A_1	$A_{1:2}$	A_3	$A_{1:4}$	A_5	$A_{5:6}$	A_7	$A_{5:8}$
A_1	$A_{1:2}$	A_3	$A_{1:4}$	A_5	$A_{5:6}$	A_7	$A_{1:8}$

[6]See Position 8 of Table 2

Table 3: **Down-sweep** for multiplying $A_1, A_2, \dots A_8$.

Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8
A_1	$A_{1:2}$	A_3	$A_{1:4}$	A_5	$A_{1:6}$	A_7	$A_{1:8}$
A_1	$A_{1:2}$	$A_{1:3}$	$A_{1:4}$	$A_{1:5}$	$A_{1:6}$	$A_{1:7}$	$A_{1:8}$

Thus, together, the up-sweep and the down-sweep of the parallel scan run in $\mathcal{O}(\log T)$ time on $\mathcal{O}(T)$ processors, and at the end of this algorithm, we get all of the intermediate products[7] (the "prefix sums").

A.4 Implementation considerations

This gentle introduction provides the main ideas and intuition of the parallel scan: closed binary associative operators leading a divide-and-conquer algorithm to leverage parallel processors to compute sequential compositions in sublinear time. However, there are also a host of implementation details for using the parallel scan when programming on accelerated hardware like GPUs (Harris et al., 2007; Yang et al., 2024; Sarnthein, 2025). For example, the presence of a general-purpose parallel scan is, as of the time of writing, a major difference between JAX (Bradbury et al., 2018) and PyTorch (Paszke et al., 2019), two leading Python libraries for deep learning. JAX has a general purpose parallel scan (`jax.lax.associative_scan`) as a fundamental primitive, which allows for implementation of a wide range of parallel scans. For example, `dynamax`, a JAX library for probabilistic state space modeling (Linderman et al., 2025), implements the parallel filtering and smoothing algorithms from Särkkä & García-Fernández (2021). In contrast, PyTorch currently has only `torch.cumsum`, which is the parallel scan where the binary associative operator is addition[8], and `torch.cumprod` (for scalar multiplication). This difference is why we implement the experiments in this paper in JAX. This lack of a general purpose parallel scan in PyTorch has also led to the custom development of highly-optimized, hardware-aware custom CUDA kernels for parallel scans, such as in Mamba (Gu & Dao, 2024), a leading SSM for language modeling. There also exist useful implementations of parallel scans for scalar/diagonal LDSs in PyTorch such as Kyrylov (2024), which can be used to implement quasi-Newton iterations in PyTorch.

B Experimental Details and Additional Experiments

We implemented our experiments using the Equinox library (Kidger & Garcia, 2021) in JAX (Bradbury et al., 2018). In our experiments that report wall-clock time, we use a stochastic implementation of quasi-Newton iterations (Zoltowski et al., 2025) that estimates the diagonal using the Hutchinson estimator (Hutchinson, 1989); see Section 3.4 of Zoltowski et al. (2025) for details.

The stopping criterion we use for deciding when the fixed-point iterations in eq. (2) have converged is based on the merit function $\mathcal{L}(\mathbf{x}_{1:T})$, which is defined as:

$$\mathbf{r}(\mathbf{x_{1:T}}) = [x_1 - f(x_0), x_2 - f(x_1), x_3 - f(x_2), \dots, x_T - f(x_{T-1})]$$
$$\mathcal{L}(\mathbf{x_{1:T}}) = \frac{1}{2}\|\mathbf{r}(\mathbf{x_{1:T}})\|_2^2.$$

In our experiments we use a tolerance of 5×10^{-4}, that is we terminate the fixed-point iterations when iterate i satisfies

$$\mathcal{L}(\mathbf{x}_{1:T}^{(i)}) \leq 5 \times 10^{-4}.$$

B.1 Experimental Details for Section 3.1

We use 10 random seeds, where the randomness controls the sequence of the S_5 group elements. Each seed uses a batch size of 16, i.e. 16 different S_5 word problems are evaluated for each run. For each seed, we

[7] See the last row of Table 3.

[8] Although Heinsen (2023) shows that clever uses of `torch.cumsum` can parallelize scalar/diagonal LDSs, of the type that are used in quasi-Newton iterations (eq. (6)).

time how long it takes for 5 runs of the fixed point solver (sequential, Picard, quasi-Newton, or Newton) to evaluate, and record this mean wall-clock time. In Figure 1, we then plot the median of these 10 mean wall-clock times. We run on an H100 with 80GB of onboard memory.

B.2 Experimental Details for Section 3.2

The experiment depicted in Figure 2 closely follows the experiments in Section 4.1 of Lim et al. (2024) and Section 6.1 Gonzalez et al. (2024). We use 10 random seeds, where the randomness controls the initialization of the GRUs, the random inputs to the GRUs, and Rademacher variables used in the stochastic implementation of quasi-Newton iterations (Zoltowski et al., 2025). The GRUs were initialized following the standard initialization practice in Equinox. Each seed uses a batch size of 16, i.e. 16 different GRU trajectories are evaluated for each run. For each seed, we time how long it takes for 5 runs of the fixed point solver (sequential, Picard, quasi-Newton, or Newton) to evaluate, and record this mean wall-clock time. In Figure 2, we then plot the median of these 10 mean wall-clock times. We run on an H100 with 80GB of onboard memory.

B.3 Experimental Details for Section 3.3

The potential ϕ used for the experiment depicted in Figure 3 is the negative log probability of a mixture of Gaussians. Each Gaussian is D-dimensional, and has a random covariance matrix drawn from a Wishart distribution. For each fixed-point method, we ran 10 random seeds, where the randomness controls the randomly chosen covariance matrices for the mixture of Gaussians and the random inputs for Langevin dynamics. Each seed uses a batch size of 16, i.e. 16 different Langevin trajectories are evaluated for each run. We use a step size $\epsilon = 1 \times 10^{-5}$ for the discrete Langevin steps. For each seed, we time how long it takes for 5 runs of the fixed point solver (sequential, Picard, quasi-Newton, or Newton) to evaluate, and record this mean wall-clock time. In Figure 3, we then plot the median of these 10 mean wall-clock times. We run on an H100 with 80GB of onboard memory. In order to get convergence for Newton iterations for longer sequence lengths, we had to run with increased precision, using the `highest` option for the `jax_default_matmul_precision` flag. See Appendix B.4 for further discussion of the importance of numerical precision for implementation of these LDS-based fixed-point methods.

We also include a small additional experiment in Figure 6: instead of varying the state size dimension, we instead vary K, the number of Gaussians that make up the potential ϕ determining the Langevin dynamics. We observe qualitatively similar results to the experiment shown in Figure 3: viz, that Picard and quasi-Newton iterations enjoy similar convergence rates and wall-clock time in settings where the Jacobian is well-approximated by the identity matrix.

B.4 Implementation Considerations

In this section, we note that while the presented fixed-point methods are parallelizable, their real-world efficiency depends on the compute environment. Choosing the appropriate method requires balancing convergence speed, computational intensity, and resource availability.

Parallel Associative Scan is Hardware-Sensitive As we have shown, the fixed-point methods discussed can be cast as LDSs and therefore be parallelized over the sequence length using a parallel associative scan. However, the practical performance of these operations depends strongly on the hardware and low-level implementation details.

For example, modern GPUs (e.g., NVIDIA A100, H100) are highly optimized for tensor operations and large batch matrix multiplications, which favor methods like Newton and quasi-Newton iterations that perform fewer, more intensive steps.

The performance gains from using more iterations of lighter-weight updates (e.g., Picard iterations) are hardware-dependent. For example, Sarnthein (2025) showed that writing custom kernels for linear recurrences as tensor operations can yield near-optimal memory-bound performance.

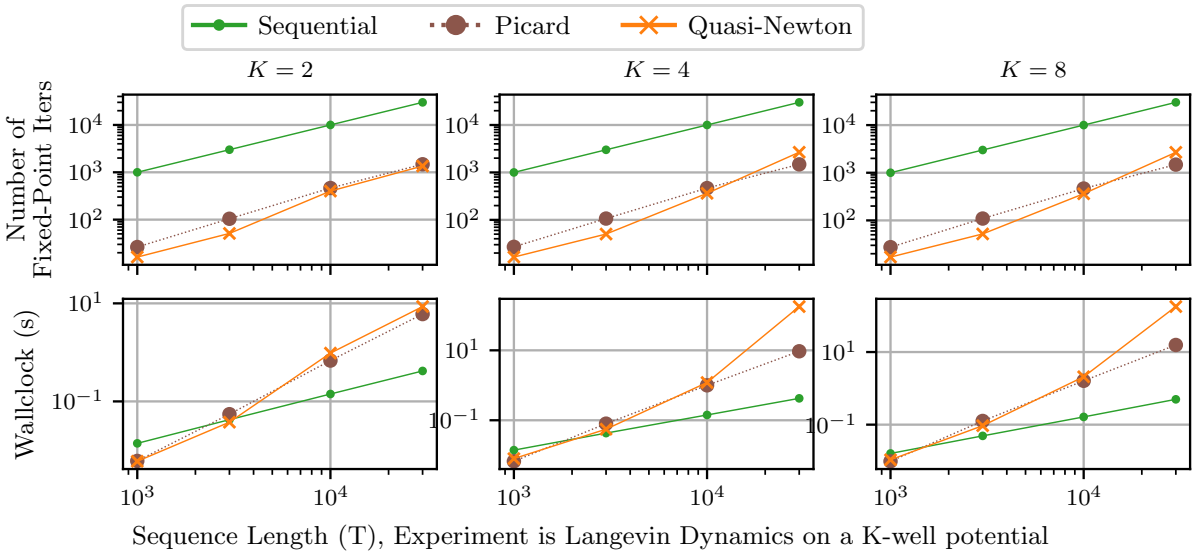

Figure 6: In a similar setting as Figure 3, we instead consider evaluating Langevin dynamics on a potential defined by the negative log probability of a mixture of K anisotropic Gaussians. Here we keep $D = 128$ throughout. We observe qualitatively similar behavior to that shown in Figure 3, where both Picard and quasi-Newton iterations enjoy similar convergence and wall-clock speed.

Memory Usage and Tradeoffs The memory requirements of different fixed-point iterations vary substantially. In particular:

- **Newton iterations** require storing and manipulating full Jacobians, which scales as $\mathcal{O}(D^2T)$ in space. This can become prohibitive for long sequences or high-dimensional hidden states.
- **Quasi-Newton iterations** reduce memory cost by using diagonal Jacobians, bringing the complexity down to $\mathcal{O}(DT)$. This is often a sweet spot for balancing memory usage and convergence rate.
- **Picard and Jacobi iterations** are the most memory-efficient, requiring only the storage of current and previous state estimates ($\mathcal{O}(DT)$), and no Jacobian-related storage.

In practice, when parallelizing over long sequences ($T \gg D$), the memory cost is often dominated by the size of intermediate state representations and the need to unroll computations over multiple fixed-point iterations. Chunking (dividing the sequence into smaller windows) and truncation (limiting the number of fixed-point iterations) are useful strategies to reduce memory usage in these settings (Dao et al., 2022; Shih et al., 2023; Selvam et al., 2024; Geiping et al., 2025; Zoltowski et al., 2025)

Numerical Stability For all fixed-point methods, numerical stability is a concern (Yaghoobi et al., 2025). In particular, LDS matrices with spectral norm close to or greater than one can cause numerical instabilities in the parallel scan operation (Gonzalez et al., 2024; 2025). This is especially critical in high-precision tasks or long sequences, and practitioners should monitor for numerical divergence or the accumulation of floating-point error.

C Extended related work

C.1 Discussion of parallel chord methods

Chapter 7 of Ortega & Rheinboldt (1970) introduces *parallel-chord methods* for solving nonlinear equations of the form $F(\mathbf{x}) = 0$ and shows that parallel-chord methods subsume both Newton and Picard iterations. A parallel-chord method $\mathcal{A}$ is an iterative process to solve for $\mathbf{x}$ satisfying $F(\mathbf{x}) = 0$ using iterations of the form

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - T^{-1} F(\mathbf{x}^{(i)}). \tag{12}$$

Newton's method corresponds to

$$T = \frac{\partial F}{\partial \mathbf{x}}.$$

The term "parallel" in this context does not have to do with applying a parallel scan over the sequence length (which we discuss at length in this paper). Instead, "parallel" in "parallel-chord methods" refers to the way in which Newton's method finds the zero of a function by making a guess for the zero, and then forming a chord that is parallel to the function that the current guess (see Figure 7). In one-dimension, the linearization is a line (a chord), while in higher-dimensions the linearization is in general a hyperplane. In Newton's method, the chord/hyperplane is tangent to the function at the current guess, while for other parallel-chord methods the approximate linearization is in general not tangent.

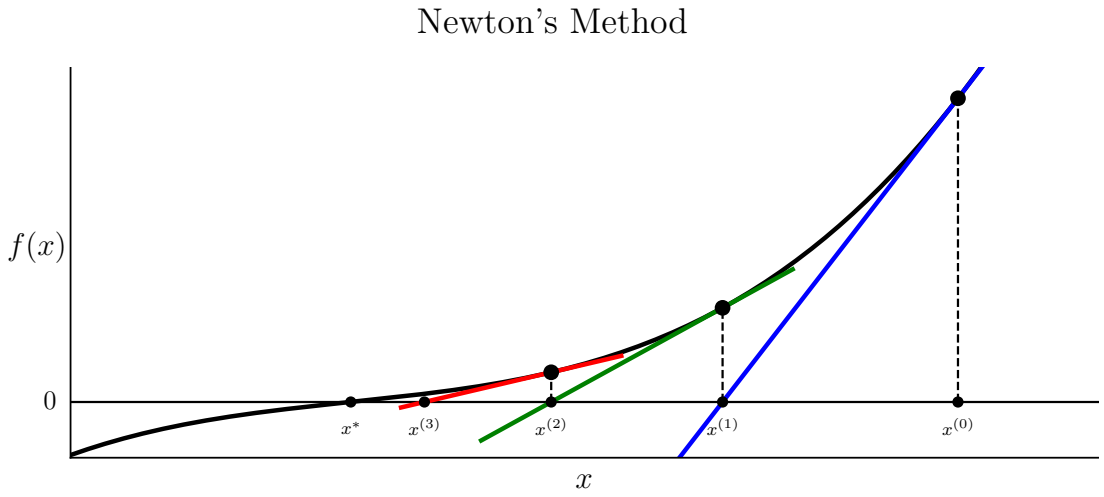

Figure 7: **The term "parallel" in parallel-chord methods.** Here we illustrate 3 iterations of Newton's method for root-finding on the one-dimensional cubic function $f(x) = (x-0.4)^3 + 0.45(x-0.4)$. We observe that each iteration of Newton's method involves forming a "parallel chord" to the function (shown in color).

While the problem $F(\mathbf{x}) = 0$ is fully general for nonlinear equations, in the context of this paper on parallelizing Markovian state space models, we consider the special setting where

$$F(\mathbf{x}) = [x_1 - f(x_0), x_2 - f(x_1), x_3 - f(x_2), \ldots, x_T - f(x_{T-1})], \tag{13}$$

and f is the nonlinear transition function, as defined in eq. (1). Thus, in the context of parallelizing sequences, it follows that

$$\frac{\partial F}{\partial \mathbf{x}} = \begin{pmatrix} I_D & 0 & 0 & \ldots & 0 & 0 \\ -\frac{\partial f_2}{\partial x}(x_1) & I_D & 0 & \ldots & 0 & 0 \\ 0 & -\frac{\partial f_3}{\partial x}(x_2) & I_D & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & I_D & 0 \\ 0 & 0 & 0 & \ldots & -\frac{\partial f_T}{\partial x}(x_{T-1}) & I_D \end{pmatrix}. \tag{14}$$

When we plug this form of the Jacobian into T in eq. (12) and simplify, we obtain the linear dynamical system in eq. (5).

In their treatment of Picard iterations, Ortega & Rheinboldt (1970) consider a more general formulation than that presented in Shih et al. (2023) or in eq. (9). Instead, similar to the definition presented in Appendix C.2.3 of Gu et al. (2021), Ortega & Rheinboldt (1970) define Picard iterations in the setting where we have removed a linear component of F, namely we have written

$$F(\mathbf{x}) =: T\mathbf{x} - G(\mathbf{x}), \tag{15}$$

for some nonsingular T. Note that such redefinition of F in terms of T and G is always possible and not uniquely determined. After making such a redefinition, Ortega & Rheinboldt (1970) define a Picard iteration as an update of the form

$$\mathbf{x}^{(i+1)} = T^{-1}G(\mathbf{x}^{(i)}). \tag{16}$$

However, by multiplying both sides of eq. (15) by T^{-1}, it follows that

$$T^{-1}G(\mathbf{x}^{(i)}) = \mathbf{x}^{(i)} - T^{-1}F(\mathbf{x}^{(i)}),$$

showing that the Picard iterations as defined in eq. (16) fit into the parallel-chord framework set out in eq. (12). Note that Picard iterations as defined by Shih et al. (2023) or in eq. (9) of this paper also fit into the framework of eq. (15): in the context of evaluating discretized ODEs, the residual defined in eq. (13) becomes

$$F_{t+1}(\mathbf{x}) = x_{t+1} - x_t - \epsilon g_t(x_t).$$

Thus, in the context of eq. (15), we have that the resulting $G_t(\mathbf{x}) = \epsilon g_{t-1}(x_{t-1})$, while the resulting T operator is given by

$$T = \begin{pmatrix} I_D & 0 & 0 & \dots & 0 & 0 \\ -I_D & I_D & 0 & \dots & 0 & 0 \\ 0 & -I_D & I_D & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I_D & 0 \\ 0 & 0 & 0 & \dots & -I_D & I_D \end{pmatrix}.$$

When we plug this T into eq. (12) and simplify, we obtain the linear dynamical system in the "Picard" row of Table 1. In general, the fixed-point methods of the common form given by eq. (4) all give rise to $T \in \mathbb{R}^{TD \times TD}$ matrices of the form

$$T = \begin{pmatrix} I_D & 0 & 0 & \dots & 0 & 0 \\ -A_2 & I_D & 0 & \dots & 0 & 0 \\ 0 & -A_3 & I_D & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I_D & 0 \\ 0 & 0 & 0 & \dots & -A_T & I_D \end{pmatrix}. \tag{17}$$

Thus, Chapter 7 of Ortega & Rheinboldt (1970) unites Newton and Picard iterations for the general root finding problem $F(\mathbf{x}) = 0$ under the umbrella of parallel-chord methods, which are iterative updates of the form of eq. (12). The framework we provide in Table 1 can be understood as a specialization of parallel-chord methods for the particular problem of sequential evaluation discussed in eq. (1). Nonetheless, we focus on how in the specific problem of sequential evaluation, which is of great interest in many areas of machine learning, a wide variety of fixed-point methods become iterative application of LDSs, allowing them to be parallelized over the sequence length with an associative scan. This important perspective about parallelizability, which is of great interest in machine learning, is not discussed in Ortega & Rheinboldt (1970) because they are considering a more general problem.

Ortega & Rheinboldt (1970) also discuss in Chapters 7 and 10 of their book how the closeness of the "parallel chord" (in general and in higher dimensions, the "approximating hyperplane") to the true linearization of

the function (Newton's method) affects the number of iterations needed for the parallel-chord method to converge. In particular, in their Section 7.1, equation 6, Ortega & Rheinboldt (1970) introduce $\sigma(\mathcal{A}, F, \mathbf{x}^*)$, which determines the rate of convergence of $\mathcal{A}$ to the solution $\mathbf{x}^*$ of $F(\mathbf{x})$. They define σ as

$$\sigma(\mathcal{A}, F, \mathbf{x}^*) := \rho\left(I - T^{-1}\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^*)\right), \tag{18}$$

where $\rho(M)$ denotes the spectral radius of a matrix M. Of course, if $T = {}^{\partial F}/_{\partial \mathbf{x}}(\mathbf{x}^*)$, then $\sigma = 0$. Thus, lower values of σ indicates that T is good approximation of the Jacobian matrix ${}^{\partial F}/_{\partial \mathbf{x}}$ evaluated at the zero $\mathbf{x}^*$ of F, while higher values of σ indicate that T is a poor approximation for ${}^{\partial F}/_{\partial \mathbf{x}}$. Ortega & Rheinboldt (1970) then use σ in their Chapter 10 (in particular, their Theorem 10.1.4) to prove linear rates of convergence for parallel-chord methods (where the rate is given by σ) within a neighborhood of the solution $\mathbf{x}^*$.

Thus, a takeaway from Ortega & Rheinboldt (1970) (as paraphrased from Gasilov et al. (1981)) is that the closer T is to ${}^{\partial F}/_{\partial \mathbf{x}}$, the fewer iterations are needed for convergence to $\mathbf{x}^*$. This takeaway is extremely similar to our guidance, though we specialize to the particular system of equations given by eq. (3) that results from the goal of rolling out the Markov process given by eq. (1). Furthermore, instead of focusing on the closeness of T to ${}^{\partial F}/_{\partial \mathbf{x}}$ (which in our setting of eq. (3) are both $TD \times TD$ matrices), we focus on the closeness of a transition matrix A_t in an LDS (see the third column of Table 1) to ${}^{\partial f}/_{\partial x}$ (both $D \times D$ matrices).

This difference in focus is important, because in the setting we consider in this paper—using fixed-point iterations of the form eq. (4) to solve nonlinear equations of the form eq. (3)—Theorem 10.1.4 of Ortega & Rheinboldt (1970) is actually *trivial*. By "trivial," we mean that it does not distinguish between the convergence rates of any of the fixed-point iterations we focus on in this paper.

To make this point more precisely, we review[9] the notion of *root-convergence*, more commonly known as *R-convergence*.

Definition 3 (R-convergence)**.** *Let $\mathcal{A}$ be a fixed-point operator with fixed-point $\mathbf{x}^*$. Let $C(\mathcal{A}, \mathbf{x}^*)$ be the set of all sequences generated by $\mathcal{A}$ which converge to $\mathbf{x}^*$. Then the R_1-factors of $\mathcal{A}$ at $\mathbf{x}^*$ are given by*

$$R_1(\mathcal{A}, \mathbf{x}^*) := \sup\left\{\limsup_{i\to\infty} \|\mathbf{x}^{(i)} - \mathbf{x}^*\|^{1/i} \;\middle|\; \{\mathbf{x}^{(i)}\}_{i\geq 0} \in C(\mathcal{A}, \mathbf{x}^*)\right\}. \tag{19}$$

Intuitively, $R_1(\mathcal{A}, \mathbf{x}^*)$ gives the rate of linear convergence of a fixed-point operator $\mathcal{A}$ to its fixed-point $\mathbf{x}^*$. Theorem 10.1.4 of Ortega & Rheinboldt (1970) implies that if $\mathcal{A}$ is a parallel-chord method, then $R_1(\mathcal{A}, \mathbf{x}^*) = \sigma(\mathcal{A}, F, \mathbf{x}^*)$. Therefore, if $\sigma > 0$, then σ is the rate of R-linear convergence of $\mathcal{A}$ to $\mathbf{x}^*$, while if $\sigma = 0$, we say that $\mathcal{A}$ converges *R-superlinearly*. However, it is important to note that these definitions are *asymptotic* in nature.

The fixed-point iterations considered in this paper, i.e. following the common form eq. (4), all have $\sigma = 0$, and therefore can be said to converge R-superlinearly.

Proposition 4. *Let $F(\mathbf{x}) = 0$ be a nonlinear equation of the form eq. (3) with solution $\mathbf{x}^*$. Let $\mathcal{A}$ be a parallel-chord method with fixed-point $\mathbf{x}^*$. Then*

$$\sigma\left(\mathcal{A}, F, \mathbf{x}^*\right) = 0.$$

Proof. As shown in eq. (14) and eq. (17), both ${}^{\partial F}/_{\partial \mathbf{x}}$ and T are lower-triangular matrices with all $D \times D$ identity matrices on their main block-diagonal. In particular, T^{-1} is also a lower-triangular matrix with all $D \times D$ identity matrices on its main block-diagonal. Consequently, the product $T^{-1}{}^{\partial F}/_{\partial \mathbf{x}}$ is also a lower-triangular matrix with all $D \times D$ identity matrices on its main block-diagonal. As a result, $I - T^{-1}{}^{\partial F}/_{\partial \mathbf{x}}$ is a lower-triangular matrix with all zeros on its main block-diagonal, and so has all its eigenvalues equal to 0. Consequently, its spectral radius is equal to zero. □

It may seem counterintuitive that even Jacobi iterations technically enjoy R-superlinear convergence in the context of parallelizing Markov processes. However, this seemingly strange result stems from the asymptotic

[9] We follow the presentation of Chapter 9 of Ortega & Rheinboldt (1970), in particular Definition 9.2.1.

nature of Definition 3 of R-convergence, and the fact that Proposition 1 of Gonzalez et al. (2024) guarantees that all fixed-point iterations of the form given by eq. (4) will converge to $\mathbf{x}^*$ in a finite number of iterations (T, to be exact). Therefore, for any LDS fixed-point scheme, we always have $\lim_{i\to\infty} \|\mathbf{x}^{(i)} - \mathbf{x}^*\| = 0$.

Because this asymptotic notion of R-convergence is not suitable in our setting, we instead use empirical case studies in Section 3 to show that the closeness of A_t to $\partial f_t/\partial x$ impacts the number of iterations needed for $\mathcal{A}$ to converge. This empirical approach also highlights how the increased computational cost of higher-order fixed-point methods affects wall-clock time on GPUs.

C.2 Other parallel fixed-point techniques

Many other fixed-point techniques can fit into the general framework we propose in Table 1 and eq. (4) (see the general algorithm in Algorithm 1). We focus on Newton, quasi-Newton, Picard, and Jacobi in the main text because of their prominence and canonicity. However, other fixed-point iterations also fit into this framework.

For example, Gonzalez et al. (2024) introduces the scale-ELK algorithm which for some $k \in [0, 1]$ sets the transition matrix A to be

$$A_t = (1 - k)\frac{\partial f_t}{\partial x}.$$

Scale-ELK can also be applied to the diagonal approximation for quasi-Newton methods. However, scale-ELK introduces an additional hyperparameter k, which is ideally chosen to keep the eigenvalues of A_t below 1 in magnitude to assure that the LDS is stable.

Therefore, we propose *clip-ELK*, which is a hyperparameter free approach to achieve the same goal of a stable LDS. Clip-ELK applies to the diagonal approximation only, and simply clips each element of A_t (which in this setting is a diagonal matrix) to be between $[-1, 1]$. Clip-ELK is immediately also a part of the framework set out in Table 1 of fixed-point methods that parallelize the evaluation of sequences via iterative application of LDS; moreover, it too must also converge globally by Proposition 1 of Gonzalez et al. (2024). An interesting and important direction for future work includes both developing and interpreting more fixed-point methods in the context of this framework that we propose.