# Combining Recurrent, Convolutional, and Continuous-time Models with Structured Learnable Linear State-Space Layers

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Recurrent neural networks (RNNs), temporal convolutions, and neural differential equations (NDEs) are popular families of deep learning models for time-series data, each with unique strengths and tradeoffs in modeling power and computational efficiency. We introduce a simple sequence model inspired by control systems that generalizes these approaches while addressing their shortcomings. The Linear State-Space Layer (LSSL) maps a sequence $u \mapsto y$ by simply simulating a linear continuous-time state-space representation $\dot{x} = Ax + Bu, y = Cx + Du$. Theoretically, we show that LSSL models are closely related to the three aforementioned families of models and inherit their strengths. For example, they generalize convolutions to continuous-time, explain common RNN heuristics, and share features of NDEs such as time-scale adaptation. We then incorporate and generalize recent theory on continuous-time memorization to introduce a trainable subset of *structured* matrices $A$ that endow LSSLs with long-range memory. Furthermore, these Structured Learnable LSSLs (SLLSSL) can even be computed asymptotically faster than the previous families of methods. Empirically, stacking SLLSSL layers into a simple deep neural network obtains state-of-the-art results across time series benchmarks for long dependencies in sequential image classification, real-world healthcare regression tasks, and speech. On a difficult speech classification task with length-16000 sequences, SLLSSL outperforms prior approaches by 24 accuracy points with 10x faster training, and even outperforms baselines that use hand-crafted features on 100x shorter sequences.

## 1 Introduction

A longstanding challenge in machine learning is efficiently modeling sequential data longer than a few thousand time steps. The usual paradigms for designing sequence models involve recurrence (e.g. RNNs), convolutions (e.g. CNNs), or differential equations (e.g. NDEs), which each come with tradeoffs. For example, RNNs are a natural stateful model for sequential data that require only constant computation/storage per time step, but are slow to train and suffer from optimization difficulties (e.g., the "vanishing gradient problem" [17]), which empirically limits their ability to handle long sequences. CNNs encode local context and enjoy fast, parallelizable training, but are not sequential, resulting in more expensive inference and an inherent limitation on the context length. NDEs are a principled mathematical model that can theoretically address continuous-time problems and long-term dependencies [16], but are very inefficient.

Ideally, a model family would combine the strengths of these paradigms, providing properties like parallelizable training (convolutional), stateful inference (recurrence) and time-scale adaptation (differential equations), while handling very long sequences in a computationally efficient way.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

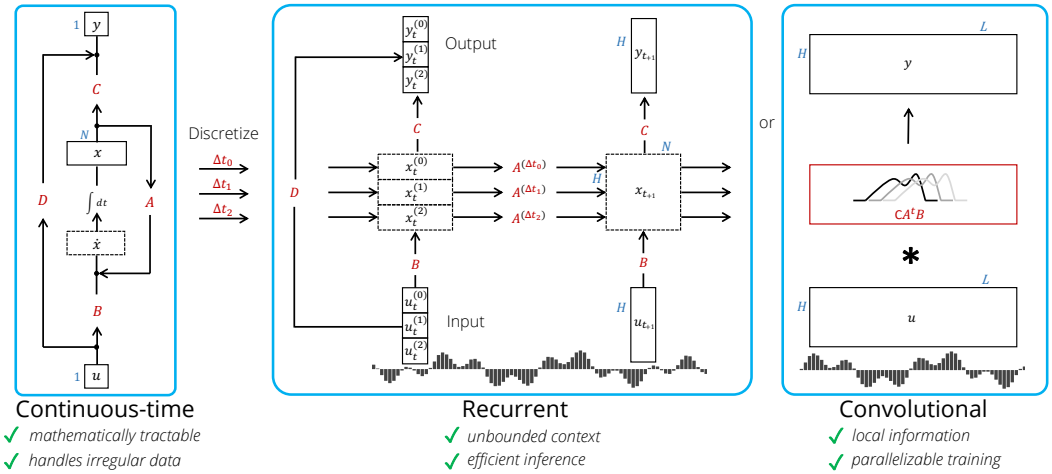


Figure 1: (**Three views of the LSSL**) A **Linear State Space Layer** layer is a map $u_t \in \mathbb{R}^H \to y_t \in \mathbb{R}^H$ defined by discretizing a state-space model $A, B, C, D$ with a parameter $\Delta t$ and stacking it $H$ times independently. The underlying state space model defines a discrete recurrence through combining the state matrix $A$ and timescale $\Delta t$ into a transition matrix $A^{(\Delta t)}$. (Parameters in red; computations in boxes; dimensionality in blue.) (**Left**) As an implicit continuous model, test-time adaptation can be performed by discretizing the same matrix $A$ using a different timescale $\Delta t$. (**Center**) As a recurrent model, inference can be performed by computing the layer *timewise* (i.e., one vertical slice at a time $(u_t, x_t, y_t), (u_{t+1}, x_{t+1}, y_{t+1}), \ldots$), by unrolling the linear recurrence. (**Right**) As a convolutional model, training can be performed by computing the layer *depthwise* (i.e., one horizontal slice at a time $(u_t)_{t\in[L]}, (y_t)_{t\in[L]}, \ldots$), in parallel by convolving with a particular filter.

Several recent works have turned to this question. These include the CKConv, which models a continuous convolution kernel [20]; several ODE-inspired RNNs, such as the UnICORNN [22]; the LMU, which speeds up a specific linear recurrence using convolutions [5, 28]; and HiPPO [10], a generalization of the LMU that introduces a theoretical framework for continuous-time memorization. However, these model families come at the price of reduced *expressivity*: intuitively, a family that is both convolutional and recurrent should be more restrictive than either.

Our first goal is to construct an expressive model family that combines all 3 paradigms while preserving their strengths. The **Linear State-Space Layer (LSSL)** is a simple sequence model that maps a 1-dimensional function or sequence $u(t) \mapsto y(t)$ through a latent state $x(t)$ by simulating a linear continuous-time state-space representation in discrete-time

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$
$$y(t) = Cx(t) + Du(t), \tag{2}$$

where $A$ controls the evolution of the system and $B, C, D$ are projection parameters. The LSSL can be viewed as an instantiation of each family, inheriting their strengths (Fig. 1):

- **LSSLs are recurrent.** If a discrete step-size $\Delta t$ is specified, the LSSL can be discretized into a linear recurrence using standard techniques, and simulated during inference as a stateful recurrent model with constant memory and computation per time step.
- **LSSLs are convolutional.** The linear time-invariant systems defined by (1)+(2) are known to be explicitly representable as a continuous convolution. Moreover, the discrete-time version can be parallelized during training using convolutions [5, 20].
- **LSSLs are continuous-time.** The LSSL itself is a differential equation. As such, it can perform unique applications of continuous-time models, such as simulating continuous processes, handling missing data [21], and adapting to different timescales.

Surprisingly, we show that LSSLs do not sacrifice expressivity, and in fact generalize convolutions and RNNs. First, classical results from control theory imply that all 1-D convolutional kernels can be approximated by an LSSL [29]. Additionally, we provide two results relating RNNs and ODEs that may be of broader interest, e.g. showing that some RNN architectural heuristics (such as gating

mechanisms) are related to the step-size $\Delta t$ and can actually be derived from ODE approximations. As corollaries of these results, we show that popular RNN methods are special cases of LSSLs.

The generality of LSSLs does come with tradeoffs. In particular, we describe and address two challenges that naive LSSL instantiations face when handling long sequences: (i) they inherit the limitations of both RNNs and CNNs at remembering long dependencies, and (ii) choosing the state matrix $A$ and timescale $\Delta t$ appropriately are critical to their performance, yet learning them is computationally infeasible. We simultaneously address these challenges by specializing LSSLs using a carefully chosen class of structured matrices $A$ (**Structured Learnable LSSLs (SLLSSL)**), such that (i) SLLSSLs generalize prior work on continuous-time memory [10] and mathematically capture long dependencies with respect to a learnable family of measures, and (ii) with new algorithms, SLLSSLs surprisingly are computationally faster than both recurrent and convolutional models, even while learning the measure $A$ and timescale $\Delta t$.

We empirically validate that SLLSSLs widely effective on benchmark datasets and very long time series from healthcare sensor data, images, and speech.

- On benchmark datasets, SLLSSLs obtain SoTA over recent RNN, CNN, and NDE-based methods across sequential image classification tasks (e.g., by over 10% accuracy on sequential CIFAR) and healthcare regression tasks with length-4000 time series (by up to 80% reduction in RMSE).
- To showcase the potential of SLLSSLs to unlock applications with extremely long sequences, we introduce a new sequential CelebA classification task with length-38000 sequences. A small SLLSSL comes within 2.16 accuracy points of a specialized ResNet-18 vision architecture which has 10x more parameters and is trained directly on images.
- Finally, we test SLLSSLs on a difficult dataset of high-resolution speech clips, where usual speech pipelines pre-process the signals to reduce the length by 100x. When training on the *raw* length-16000 signals, the SLLSSL not only (i) outperforms previous methods by over 20 accuracy points in 1/5 the training time, but (ii) sets a true SoTA, even compared to baselines that use the pre-processed length-160 sequences, overcoming the limitations of hand-crafted feature engineering.

**Summary of Contributions**

- We introduce Linear State-Space Layers (LSSLs), a simple sequence-to-sequence transformation that shares the modeling advantages of recurrent, convolutional, and continuous-time methods. Conversely, we prove that RNNs and CNNs can be seen as special cases of LSSLs (Section 3).
- We prove that a structured subclass of LSSLs can learn representations that solve continuous-time memorization, allowing it to adapt its measure and timescale (Section 4.1). We also provide fast algorithms for these Structured Learnable LSSLs (SLLSSLs), showing that they are computationally more efficient than previous models at both training and inference (Section 4.2).
- Empirically, we show that SLLSSLs stacked into a deep neural network are widely effective on time series data, even (or *especially*) on extremely long sequences (Section 5).

## 2 Technical Background

We summarize the preliminaries on differential equations that are necessary for this work. We first introduce two standard approximation schemes for differential equations that we will use to convert continuous-time models to discrete-time, and will be used in our results on understanding RNNs. We then give an overview of the HiPPO framework for continuous-time memorization [10], which will give us a mathematical tool for constructing LSSLs that can address long-term dependencies.

**Approximations of differential equations.** Any differential equation $\dot{x}(t) = f(t, x(t))$ has an equivalent *integral equation* $x(t) = x(t_0) + \int_{t_0}^{t} f(s, x(s))\, ds$. This can be numerically solved by storing some approximation for $x$, and keeping it fixed inside $f(t, x)$ while iterating the equation. For example, *Picard iteration* is often used to prove the existence of solutions to ODEs by iterating the equation $x_{i+1}(t) := x_i(t_0) + \int_{t_0}^{t} f(s, x_i(s))\, ds$ . In other words, it finds a sequence of functions $x_0(t), x_1(t), \ldots$ that approximate the solution $x(t)$ of the integral equation.

**Discretization.** On the other hand, for a desired sequence of discrete times $t_i$, approximations to $x(t_0), x(t_1), \ldots$ can be found by iterating the equation $x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(s, x(s))\, ds$.

Different ways of approximating the RHS integral lead to different discretization schemes. We single out a discretization method called the **generalized bilinear transform (GBT)** which is specialized to linear ODEs of the form (1). Given a *step size* $\Delta t$, the GBT update is

$$x(t+\Delta t) = (I - \alpha \Delta t \cdot A)^{-1}(I + (1-\alpha)\Delta t \cdot A)x(t) + \Delta t(I - \alpha \Delta t \cdot A)^{-1} B \cdot u(t).\Delta t \quad (3)$$

Three important cases are: $\alpha = 0$ becomes the classic *Euler method* which is simply the first-order approximation $x(t+\Delta t) = x(t) + \Delta t \cdot x'(t)$; $\alpha = 1$ is called the *backward Euler* method; and $\alpha = \frac{1}{2}$ is called the *bilinear* method, which preserves the stability of the system [30].

In Section 3.2 we will show that the backward Euler method and Picard iteration are actually related to RNNs. On the other hand, the bilinear discretization will be our main method for computing accurate discrete-time approximations of our continuous-time models. In particular, define $A^{(\Delta t)}$ and $B^{(\Delta t)}$ to be the matrices appearing in (3) for $\alpha = \frac{1}{2}$. Then the discrete-time state-space model is

$$x_t = A^{(\Delta t)} x_{t-1} + B^{(\Delta t)} u_t \quad (4)$$
$$y_t = C x_t + D u_t. \quad (5)$$

**$\Delta t$ as a timescale.** In most models, the length of dependencies they can capture is roughly proportional to $\frac{1}{\Delta t}$. Thus we also refer to the step size $\Delta t$ as a *timescale*. This is an intrinsic part of converting a continuous-time ODE into a discrete-time recurrence, and most ODE-based RNN models have it as an important and non-trainable hyperparameter [10, 22, 28]. On the other hand, in Section 3.2 we show that the gating mechanism of classical RNNs is a version of learning $\Delta t$. Ideally, all ODE-based discrete models would be able to automatically learn the proper timescales.

**Continuous-time memory.** Consider an input function $u(t)$, a fixed probability measure $\omega(t)$, and a sequence of $N$ basis functions such as polynomials. At every time $t$, the history of $u$ before time $t$ can be projected onto this basis, which yields a vector of coefficients $x(t) \in \mathbb{R}^N$ that represents an optimal approximation of the history of $u$ with respect to the provided measure $\omega$. The map taking the function $u(t) \in \mathbb{R}$ to coefficients $x(t) \in \mathbb{R}^N$ is called the **High-Order Polynomial Projection Operator (HiPPO)** with respect to the measure $\omega$. In special cases such as the uniform measure $\omega = \mathbb{I}\{[0,1]\}$ and the exponentially-decaying measure $\omega(t) = \exp(-t)$, Gu et al. [10] showed that $x(t)$ satisfies a differential equation $\dot{x}(t) = A(t)x(t) + B(t)u(t)$ and derived closed forms for the matrix $A$. Their framework provides a principled way to design memory models handling long dependencies; However, they prove only these few special cases.

# 3 Linear State-Space Layers (LSSL)

We define our main abstraction, a model family that generalizes recurrence and convolutions. Section 3.1 first formally defines the LSSL, then discusses how to compute it with multiple views. Conversely, Section 3.2 shows that LSSLs capture convolutions and the most popular RNNs.

## 3.1 Different Views of the LSSL

Given a fixed state space representation $A, B, C, D$, an LSSL is the sequence-to-sequence mapping defined by discretizing the linear state-space model (1) and (2) on $H$ independent input features.

Concretely, an LSSL layer has parameters $A, B, C, D$, and $\Delta t$. It operates on an input $u \in \mathbb{R}^{L \times H}$ representing a sequence of length $L$ where each timestep has an $H$-dimensional feature vector. Each feature $h \in [H]$ defines a sequence $(u_t^{(h)})_{t \in [L]}$, which is combined with a timescale $\Delta t_h$ to define an output $y^{(h)} \in \mathbb{R}^L$ via the discretized state-space model (4)+(5).

Computationally, the discrete-time LSSL can be viewed in multiple ways (Fig. 1).

**As a recurrence.** The recurrent state $x_{t-1} \in \mathbb{R}^{H \times N}$ carries the context of all inputs before time $t$. The current state $x_t$ and output $y_t$ can be computed by simply following equations (4)+(5). Thus the LSSL is a recurrent model with efficient and stateful inference, which can consume a (potentially unbounded) sequence of inputs while requiring fixed computation/storage per time step.

**As a convolution.** For simplicity let the initial state be $x_{-1} = 0$. Then (4)+(5) explicitly yields

$$y_k = C\left(A^{(\Delta t)}\right)^k B^{(\Delta t)} u_0 + C\left(A^{(\Delta t)}\right)^{k-1} B^{(\Delta t)} u_1 + \cdots + C A^{(\Delta t)} B^{(\Delta t)} u_{k-1} + B^{(\Delta t)} u_k + D u_k. \quad (6)$$

Then $y$ is simply the (non-circular) convolution $y = \mathcal{K}_L(A^{(\Delta t)}, B^{(\Delta t)}, C) * u + Du$, where

$$\mathcal{K}_L(A, B, C) = \left(CA^iB\right)_{i\in[L]} \in \mathbb{R}^L = (CB, CAB, \ldots, CA^{L-1}B). \tag{7}$$

Thus the LSSL can be viewed as a convolutional model where the entire output $y \in \mathbb{R}^{H\times L}$ can be computed at once by a convolution, which can be efficiently implemented with three FFTs.

**The computational bottleneck.** We make a note that the bottleneck of (i) the recurrence view is **matrix-vector multiplication (MVM)** by the discretized state matrix $A^{(\Delta t)}$ when simulating (4), and (ii) the convolutional view is computing the **Krylov function** $\mathcal{K}_L$ (7). Throughout this section we assumed the LSSL parameters were fixed, which means that $A^{(\Delta t)}$ and $\mathcal{K}_L(A, B, C)$ can be cached for efficiency. However, learning the parameters $A^{(\Delta t)}$ and $\Delta t$ would involve repeatedly re-computing these, which is infeasible in practice. We revisit and solve this problem in Section 4.2.

### 3.2 Expressivity of LSSLs

For a model to be both recurrent and convolutional, one might expect it to be limited in other ways. Indeed, while [5, 20] also observe that certain recurrences can be replaced with a convolution, they note that it is not obvious if convolutions can be replaced by recurrences. Moreover, while the LSSL is a linear recurrence, popular RNN models are *nonlinear* sequence models with activation functions between each time step. We now show that LSSLs surprisingly do not have limited expressivity.

**Convolutions are LSSLs.** A well-known fact about state-space systems (1)+(2) is that the output $y$ is related to the input $u$ by a convolution $y(t) = \int h(\tau)u(t-\tau)d\tau$ with the *impulse response* $h$ of the system. Conversely, a convolutional filter $h$ that is a rational function of degree $N$ can be represented by a state-space model of size $N$ [29]. Thus, an arbitrary convolutional filter $h$ can be approximated by a rational function (e.g., by Padé approximants) and represented by an LSSL.

**RNNs are LSSLs.** We show two results about RNNs that may be of broader interest. Our first result says that the ubiquitous *gating mechanism* of RNNs, commonly perceived as a heuristic to smooth optimization [12], is actually the analog of a step size or timescale $\Delta t$.

**Lemma 3.1.** *A (1-D) gated recurrence $x_t = (1-\sigma(z))x_{t-1} + \sigma(z)u_t$, where $\sigma$ is the sigmoid function and $z$ is an arbitrary expression, can be viewed as the GBT($\alpha = 1$) (i.e.,* backwards-Euler*) discretization of a 1-D linear ODE $\dot{x}(t) = -x(t) + u(t)$.*

*Proof.* Applying a discretization requires a positive step size $\Delta t$. The simplest way to parameterize a positive function is via the exponential function $\Delta t = \exp(z)$ applied to any expression $z$. Substituting this into (3) with $A = -1, B = 1, \alpha = 1$ exactly produces the gated recurrence. □

While Lemma 3.1 involves approximating continuous systems using discretization, the second result is about approximating them using Picard iteration (Section 2). Roughly speaking, each layer of a deep *linear* RNN can be viewed as successive Picard iterates $x_0(t), x_1(t), \ldots$ approximating a function $x(t)$ defined by a *non-linear* ODE. This shows that we do not lose modeling power by using linear instead of non-linear recurrences, and that the nonlinearity can instead be "moved" to the depth direction of deep neural networks to improve speed without sacrificing expressivity.

**Lemma 3.2.** *(Infinitely) deep stacked LSSL layers of order $N = 1$ with* position-wise *non-linear functions can approximate any non-linear ODE $\dot{x}(t) = -x + f(t, x(t))$.*

We note that many of the most popular and effective RNN variants such as the LSTM [12], GRU [6], QRNN [4], and SRU [14], involve a hidden state $x_t \in \mathbb{R}^H$ that involves independently "gating" the $H$ hidden units. Applying Lemma 3.1, they actually also approximate an ODE of the form in Lemma 3.2. Thus LSSLs and these popular RNN models can be seen to all approximate the same type of underlying continuous dynamics, by using Picard approximations in the depth direction and discretization (gates) in the time direction. Appendix B gives precise statements and proofs.

## 4 Structured Learnable LSSLs: Fast LSSLs with Continuous-time Memory

In Section 3 we introduced the LSSL model and showed that it shares the strengths of convolutions and recurrences while also generalizing them. We now discuss and address its main limitations,

Table 1: Complexity of various sequence models in terms of length (**L**), batch size (**B**), and hidden dimension (**H**); tildes denote log factors. Metrics are parameter count, training computation, training memory requirement, training parallelization, and inference computation (for 1 sample and time-step). For simplicity, the state size $N$ of LSSLs is tied to $H$. Bold denotes model is best for that metric. Derivations and more results in Appendix C.

| | Convolution | RNN | LSSL | SLLSSL |
|---|---|---|---|---|
| Parameters | $LH^2$ | $\mathbf{H^2}$ | $\mathbf{H^2}$ | $\mathbf{H^2}$ |
| Training | $BLH^2 + \tilde{L}H(B+H)$ | $BLH^2$ | $H^2(H^2 + LH + B\tilde{L})$ | $\mathbf{BH(\tilde{H} + \tilde{L}) + B\tilde{L}H}$ |
| Memory | $BLH + LH^2$ | $\mathbf{BLH}$ | $H^3 + LH^2 + BLH$ | $\mathbf{BLH}$ |
| Parallel | **Yes** | No | **Yes** | **Yes** |
| Inference | $LH^2$ | $\mathbf{H^2}$ | $H^3$ | $\mathbf{H^2}$ |

in particular handling long dependencies (Section 4.1) and efficient computation (Section 4.2). by introducing a particular class of structured matrices $A$ that simultaneously overcomes both challenges.

### 4.1 Incorporating Memory into LSSLs (MLSSL)

The generality of LSSLs means they can inherit the issues of recurrences and convolutions at addressing long dependencies (Section 1). For example, viewed as a recurrence, repeated multiplication by $A^{(\Delta t)}$ could suffer from the *vanishing gradients* problem [17, 20]. We confirm empirically that LSSLs with random state matrices $A$ are not effective (Section 5.4).

However, one advantage of these continuous-time, mathematical models is that they are theoretically analyzable, and specific $A$ matrices can be derived to address this issue. In particular, the HiPPO framework (Section 2) describes how to memorize a function in continuous time with respect to a measure $\omega$ [10]. This operator mapping a function to a continuous representation of its past is denoted $\mathrm{hippo}(\omega)$, and was shown to have the form of equation (1) in three special cases. However, these matrices are non-trainable in the sense that no other $A$ matrices were known to be $\mathrm{hippo}$ operators.

To address this, we theoretically resolve the open question from [10], showing that $\mathrm{hippo}(\omega)$ for *any measure* $\omega$ [1] results in (1) with a structured matrix $A$.

**Theorem 1** (Informal)**.** *For an arbitrary measure $\omega$, the optimal memorization operator* $\mathrm{hippo}(\omega)$ *has the form* $\dot{x}(t) = Ax(t) + Bu(t)$ (1) *for a* low recurrence-width (LRW) *[8] state matrix $A$.*

For measures covering the classical orthogonal polynomials (OPs) [25], there is even more structure.

**Corollary 4.1.** *For $\omega$ corresponding to the classical OPs,* $\mathrm{hippo}(\omega)$ *is* 3*-quasiseparable.*

Although beyond the scope of this section, we mention that LRW matrices are a type of structured matrix that have linear MVM [8]. In Appendix C we define this class and prove Theorem 1.

Theorem 1 tells us that a LSSL that uses a state matrix $A$ within a particular class of structured matrices would carry the theoretical interpretation of continuous-time memorization. Ideally, we would be able to automatically learn the best $A$ within this class; however, this runs into computational challenges which we address next (Section 4.2). For now, we define the **Memory LSSL (MLSSL)** to be one where the $A$ matrix is fixed to one of the HiPPO matrices prescribed by [10].

### 4.2 Computational Efficiency with the Structured Learnable LSSL (SLLSSL)

Although $A$ and $\Delta t$ are the most critical parameters of an LSSL which govern the state-space (c.f. Section 4.1) and timescale (Sections 2 and 3.2), they are not feasible to train in a naive LSSL. In particular, Section 3.1 noted that it would require efficient *matrix-vector multiplication (MVM)* and *Krylov function* (7) for $A^{(\Delta t)}$ to compute the recurrent and convolutional views, respectively. However, the former seems to involve a matrix inversion (3), while the latter seems to require powering $A^{(\Delta t)}$ up $L$ times.

In this section, we show that the same restriction of $A$ to the class of quasiseparable (Corollary 4.1), which gives an LSSL the ability to theoretically remember long dependencies, simultaneously grants it computational efficiency.

[1] To be precise, the measures that correspond to orthogonal polynomials [25].

First of all, it is known that quasiseparable matrices have efficient (linear-time) MVM [18]. We show that they also have fast Krylov functions, allowing efficient training with convolutions.

**Theorem 2.** *For any $k$-quasiseparable matrix $A$ (with constant $k$) and arbitrary $B, C$, the Krylov function $\mathcal{K}_L(A, B, C)$ can be computed in* quasi-linear *time and storage $\tilde{O}(N + L)$ and* logarithmic *depth (i.e., is parallelizable).*

We remark that Theorem 2 is non-obvious. To illustrate, it is easy to see that unrolling (7) for a general matrix $A$ takes time $LN^2$. Even if $A$ is extremely structured with linear computation, it requires $LN$ operations and linear depth. The depth can be reduced with the squaring technique (batch multiply by $A, A^2, A^4, \ldots$), but this then requires $LN$ intermediate storage. In fact, the algorithm for Theorem 2 is quite elegant (Appendix C) and involves a divide-and-conquer recursion over matrices of *polynomials*, using the observation that (7) is related to the power series $C(I - Ax)^{-1}B$ .

An **SLLSSL** then refers to a LSSL with $A$ satisfying Corollary 4.1. Table 1 summarizes and compares various measures of complexity for the models we consider. In conclusion, learning within this structured matrix family simultaneously endows LSSLs with long-range memory through Theorem 1 and is computationally feasible (in fact, faster than baselines) through Theorem 2.

# 5 Empirical Evaluation

We test SLLSSLs empirically on a range of time series datasets with sequences from length 160 up to 38000 (Sections 5.1 and 5.2), where they substantially improve over prior work. We additionally validate the computational and modeling benefits of LSSLs from generalizing all three main model families (Section 5.3), and analyze the benefits of incorporating principled memory representations that can be learned (Section 5.4).

**Baselines.** Our tasks have extensive prior work and we evaluate against previously reported best results. We highlight our primary baselines, three very recent works explicitly designed for long sequences: CKConv (a continuous-time CNN) [20], UnICORNN (an ODE-inspired RNN) [22], and Neural Controlled/Rough Differential Equations (NCDE/NRDE) (a sophisticated NDE) [13, 16]. These are the only models we are aware of that have experimented with sequences of length >10k.

## 5.1 State-of-the-Art on Long Time Series Benchmarks

We test on the sequential MNIST, permuted MNIST, and sequential CIFAR tasks (Table 2), popular benchmarks which were originally designed to test the ability of recurrent models to capture long-term dependencies of length up to 1k [1]. SLLSSL sets SoTA on sCIFAR by more than 10 points. We note that all results were achieved with at least 5x fewer parameters than the previous SoTA (Appendix D).

Table 2: (**Pixel-by-pixel image classification.**) (Top) our methods. (Middle) recurrent baselines. (Bottom) convolutional + other baselines.

| Model | sMNIST | pMNIST | sCIFAR |
|---|---|---|---|
| **SLLSSL** | **99.53** | **98.76** | **84.65** |
| **MLSSL** | **99.50** | **98.60** | **81.97** |
| LMUFFT [5] | - | 98.49 | - |
| UNIcoRNN [22] | - | 98.4 | - |
| HiPPO-RNN [10] | 98.9 | 98.3 | 61.1 |
| URGRU [11] | 99.27 | 96.51 | 74.4 |
| r-LSTM [27] | 98.4 | 95.2 | 72.2 |
| LSTM [12][11] | 98.9 | 95.11 | 63.01 |
| CKConv [20] | 99.32 | 98.54 | 63.74 |
| TrellisNet [3] | 99.20 | 98.13 | 73.42 |
| TCN [2] | 99.0 | 97.2 | - |
| Transformer [27] | 98.9 | 97.9 | 62.2 |

Table 3: (**Vital signs prediction.**) RMSE for predicting respiratory rate (RR), heart rate (HR), and blood oxygen (SpO2). * indicates our own runs to complete results for the strongest baselines.

| Model | RR | HR | SpO2 |
|---|---|---|---|
| **SLLSSL** | **0.350** | **0.432** | **0.141** |
| **MLSSL** | **0.378** | **0.561** | **0.221** |
| UnICORNN [22] | 1.06 | 1.39 | 0.869* |
| CKConv | 1.214* | 2.05* | 1.051* |
| NRDE [16] | 1.49 | 2.97 | 1.29 |
| IndRNN [22] | 1.47 | 2.1 | - |
| expRNN [22] | 1.57 | 1.87 | - |
| Transformer | 2.61* | 12.2* | 3.02* |
| XGBoost [26] | 1.67 | 4.72 | 1.52 |
| Random Forest [26] | 1.85 | 5.69 | 1.74 |
| Ridge Regress. [26] | 3.86 | 17.3 | 4.16 |

We additionally use the BDIMC healthcare datasets (Table 3), a suite of widely studied time series regression problems of length 4000 on estimating vital signs. SLLSSL reduces RMSE by more than two-thirds on all datasets.

### 5.2 Speech and Image Classification for *Very* Long Time Series

Raw speech is challenging for ML models due to high-frequency sampling resulting in very long sequences. Traditional systems involve complex pipelines that require feeding mixed-and-matched hand-crafted features into DNNs [19]. Table 5 reports results for the Speech Commands (SC) dataset [13] for classification of 1-second audio clips. Very few methods have made progress on the raw speech signal, instead requiring pre-processing with standard mel-frequency cepstrum coefficients (MFCC). By contrast, SLLSSL sets SoTA on this dataset *while training on the raw signal*. We note that MFCC extracts sliding window frequency coefficients and thus is related to the coefficients $x(t)$ defined by MLSSL (Section 2, Section 4.1, [10], Appendix C). Consequently, SLLSSL may be interpreted as automatically learning MFCC-type features in a trainable basis.

Table 4: (**Sequential CelebA Classification**.) Accuracy on facial attribute classification with 38000-length sequences (average gap $2.16\%$).

| | **MLSSL** | **ResNet-18** |
|---|---|---|
| **Att.** | 78.89 | 81.35 |
| **MSO** | 92.36 | 93.92 |
| **Smil.** | 90.95 | 92.89 |
| **WL** | 90.57 | 93.25 |

To stress-test LSSLs' ability to handle extremely long sequences we create a challenging new sequential-CelebA task, where we classify $178 \times 218$ images = 38000-length sequences for 4 facial attributes: Attractive (Att.), Mouth Slightly Open (MSO), Smiling (Smil.), Wearing Lipstick (WL) [15]. We chose the 4 most class-balanced attributes to avoid well-known problems with class imbalance. MLSSL comes close to matching the performance of a specialized ResNet-18 image classification architecture that has $10\times$ the parameters (Table 4). We emphasize we are the first to demonstrate that this is possible to do with a generic sequence model.

### 5.3 Advantages of Recurrent, Convolutional, and Continuous-time Models

We validate that the generality of LSSLs endows it with the strengths of all three families.

**Convergence Speed.** As a recurrent and NDE model that incorporates new theory for continuous-time memory (Section 4.1), the LSSL has strong inductive bias for sequential data, and converges rapidly to SoTA results on our benchmarks. With its convolutional view, training can be parallelized and it is also computationally efficient in practice. Table 6 compares the time it takes the MLSSL to achieve SoTA, in either sample (measured by epochs) or computational (measured by wall clock) complexity. In all cases, LSSLs reached the target in a fraction of the time of the previous model.

**Timescale Adaptation.** Table 5 also reports the results of continuous-time models that are able to handle unique settings such as missing data in time series, or test-time shift in timescale (we note that this is a realistic problem, e.g., when deployed healthcare models are tested on EEG signals that are sampled at a different rate [23, 24]). We note that many of these baselines were custom designed for such settings, which is of independent interest. On the other hand, LSSLs perform timescale adaptation by simply changing its $\Delta t$ values at inference time, while still outperforming the performance of prior methods with no shift. Additional results on the CharacterTrajectories dataset from prior work [13, 20] are in Appendix D, where SLLSSL is competitive with the best baselines.

Table 5: (**Raw Speech Classification; Timescale Shift**.) (Top): Raw signals (length 16000); $1 \to f$ indicates test-time change in sampling rate by a factor of $f$. (Bottom): Pre-processed MFCC features used in prior work (length 161). ✗ denotes computationally infeasible.

| | **SLLSSL** | **MLSSL** | CKConv | UnICORNN | N(C/R)DE | ODE-RNN [21] | GRU-ODE [7] |
|---|---|---|---|---|---|---|---|
| $1 \to 1$ | **95.87** | 90.64 | 71.66 | 11.02 | 16.49 | ✗ | ✗ |
| $1 \to \frac{1}{2}$ | **88.66** | 78.01 | 65.96 | 11.07 | 15.12 | ✗ | ✗ |
| MFCC | 93.58 | 92.55 | **95.3** | 90.64 | 89.8 | 65.9 | 47.9 |

Table 6: (**Modeling and Computational Benefits of LSSLs.**) In each benchmark category, we compare the number of epochs (ep.) it takes a MLSSL to reach the previous SoTA (PSoTA) results as well as a near-SoTA target. We also report the wall clock time it took to reach PSoTA relative to the previous best model.

| | Permuted MNIST | | | BDIMC Heart Rate | | | Speech Commands RAW | | |
|---|---|---|---|---|---|---|---|---|---|
| | 98% Acc. | PSoTA | Time | 1.5 RMSE | PSoTA | Time | 65% Acc. | PSoTA | Time |
| **MLSSL** | 16 ep. | 104 ep. | 0.19× | 9 ep. | 10 ep. | 0.07× | 9 ep. | 10 ep. | 0.14× |
| CKConv | 118 ep. | 200 ep. | 1.0× | ✗ | ✗ | ✗ | 188 ep. | 280 ep. | 1.0× |
| UnICORNN | 75 ep. | ✗ | ✗ | 116 ep. | 467 ep. | 1.0× | ✗ | ✗ | ✗ |

### 5.4 SLLSSL Ablations: Learning the Memory Dynamics and Timescale

We demonstrate that the $\Delta t$ and $A$ parameters, which SLLSSLs are able to automatically learn in contrast to prior work, are indeed critical to the performance of these continuous-time models. We note that learning $\Delta t$ adds only $O(H)$ parameters and learning $A$ adds $O(N)$ parameters, adding less than 1% parameter count compared to the base models with $O(HN)$ parameters.

**Memory dynamics $A$.** We validate that vanilla LSSLs suffer from the modeling issues described in Section 4. We tested that LSSLs with *random* $A$ matrices (normalized appropriately) perform very poorly (e.g., 62% on pMNIST). Further, we note the consistent increase in performance from MLSSL to SLLSSL despite the negligible parameter difference. These ablations show that (i) incorporating the theory of Theorem 1 is actually *necessary* for LSSLs, and (ii) further training the structured $A$ is additionally helpful, which can be interpreted as learning the measure for memorization (Section 4.1).

**Timescale $\Delta t$.** Section 3.2 showed that SLLSSL's ability to learn $\Delta t$ is its direct generalization of the critical *gating mechanism* of popular RNNs, which previous ODE-based RNN models [5, 10, 22, 28] cannot learn. We note that on sCIFAR, MLSSL with poorly-specified $\Delta t$ gets only $49.3\%$ accuracy. Additional results in Appendix D show that learning $\Delta t$ alone provides an orthogonal boost to learning $A$, and visualizes the noticeable change in $\Delta t$ over the course of training.

## 6 Discussion

In this work we introduced a simple and principled model (LSSL) inspired by a fundamental representation of physical systems. We showed theoretically and empirically that it generalizes and inherits the strengths of the main families of modern time series models, that its main limitations of long-term memory and computation can be simultaneously resolved with new theory and algorithms (MLSSL, SLLSSL), and that it is empirically effective on difficult tasks with very long sequences.

**Model simplicity.** Our models are very simple, consisting of identical L(*inear*)SSL layers with simple position-wise non-linear modules between layers (Appendix A). Our models were able to train at much higher learning rates than baselines and were not sensitive to hyperparameters, of which we did light tuning primarily on learning rate and dropout. In contrast to most baselines [3, 13, 20], we did not use hyperparameters for improving stability and regularization such as weight decay, gradient clipping, weight norm, input dropout, etc. While the most competitive recent works introduce at least one hyperparameter of critical importance (e.g. depth and step size [16], $\alpha$ and $\Delta t$ [22], $\omega_0$ [20]) that are difficult to tune, MLSSL has only $\Delta t$, which SLLSSL can even learn automatically.

**Limitations.** Theorem 2's algorithm is sophisticated (Appendix C) and we did not implement it yet, but we do not believe there is a fundamental issue. As such, the SLLSSL is slower and less memory efficient than the MLSSL in our current implementation.

**Discussion and future work.** Models struggle in applications with very long temporal data such as speech, videos, and medical time-series. We hope that our conceptual and technical contributions can unlock new applications with simple, principled, and less engineered models. We note that our pixel-level image classification experiments, which use no heuristics (batch norm, auxiliary losses) or extra information (data augmentation), perform similar to early convnet models with vastly more parameters, and is in the spirit of recent attempts at unifying data modalities with a generic sequence model [9]. Our speech results demonstrate the possibility of *learning* better features than hand-crafted processing pipelines used widely in speech applications. We are excited about potential downstream applications, such as training other downstream models on top of pre-trained LSSL features.

# References


[1] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *The International Conference on Machine Learning (ICML)*, pages 1120–1128, 2016.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *The International Conference on Learning Representations (ICLR)*, 2019.

[4] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

[5] Narsimha Chilkuri and Chris Eliasmith. Parallelizing legendre memory unit training. *arXiv preprint arXiv:2102.11417*, 2021.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *arXiv preprint arXiv:1905.12374*, 2019.

[8] Christopher De Sa, Albert Gu, Rohan Puttagunta, Christopher Ré, and Atri Rudra. A two-pronged progress in structured dense matrix vector multiplication. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1060–1079. SIAM, 2018.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[10] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *arXiv preprint arXiv:2008.07669*, 2020.

[11] Albert Gu, Caglar Gulcehre, Tom Le Paine, Matt Hoffman, and Razvan Pascanu. Improving the gating mechanism of recurrent neural networks. In *The International Conference on Machine Learning (ICML)*, 2020.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[13] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.

[14] Tao Lei, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. *arXiv preprint arXiv:1709.02755*, 2017.

[15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. In *ICCV*.

[16] James Morrill, Cristopher Salvi, Patrick Kidger, James Foster, and Terry Lyons. Neural rough differential equations for long time series. *arXiv preprint arXiv:2009.08295*, 2020.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[18] Clément Pernet. Computing with quasiseparable matrices. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 389–396, 2016.

[19] M. Ravanelli, T. Parcollet, and Y. Bengio. The pytorch-kaldi speech recognition toolkit. In *In Proc. of ICASSP*, 2019.

[20] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.

[21] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

[22] T Konstantin Rusch and Siddhartha Mishra. Unicornn: A recurrent model for learning very long time dependencies. *arXiv preprint arXiv:2103.05487*, 2021.

[23] Khaled Saab, Jared Dunnmon, Christopher Ré, Daniel Rubin, and Christopher Lee-Messer. Weak supervision as an efficient approach for automated seizure detection in electroencephalography. *NPJ Digital Medicine*, 3(1):1–12, 2020.

[24] Vinit Shah, Eva Von Weltin, Silvia Lopez, James Riley McHugh, Lillian Veloso, Meysam Golmohammadi, Iyad Obeid, and Joseph Picone. The Temple University hospital seizure detection corpus. *Frontiers in neuroinformatics*, 12:83, 2018.

[25] G. Szegö. *Orthogonal Polynomials*. Number v.23 in American Mathematical Society colloquium publications. American Mathematical Society, 1967. ISBN 9780821889527.

[26] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb. Time series extrinsic regression. *Data Mining and Knowledge Discovery*, pages 1–29, 2021. doi: https://doi.org/10.1007/s10618-021-00745-9.

[27] Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in RNNs with auxiliary losses. In *The International Conference on Machine Learning (ICML)*, 2018.

[28] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 15544–15553, 2019.

[29] Robert L Williams, Douglas A Lawrence, et al. *Linear state-space control systems*. Wiley Online Library, 2007.

[30] Guofeng Zhang, Tongwen Chen, and Xiang Chen. Performance recovery in digital implementation of analogue systems. *SIAM journal on control and optimization*, 45(6):2207–2223, 2007.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See (Section 6)
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See (Section 6)
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See (Appendix B)
   (b) Did you include complete proofs of all theoretical results? [Yes] See (Appendix B)
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]