Neural Networks 193 (2026) 108039

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Review

State-space modeling in long sequence processing: a survey on recurrence in the transformer era

Matteo Tiezzi [a,*], Michele Casoni [b], Alessandro Betti [c], Marco Gori [b], Stefano Melacci [b]

[a] IIT, Ist. Italiano di Tecnologia, Genova, 16152, Italy
[b] DIISM, University of Siena, 53100, Siena, Italy
[c] IMT, Scuola Alti Studi, 55100, Lucca, Italy

ARTICLE INFO

Keywords:
Recurrent neural networks
Transformers
State-Space models
Long sequences
Learning over time

ABSTRACT

Effectively learning from sequential data is a longstanding goal of Artificial Intelligence, especially in the case of long sequences. From the dawn of Machine Learning, several researchers have pursued algorithms and architectures capable of processing sequences of patterns, retaining information about past inputs while still leveraging future data, without losing precious long-term dependencies and correlations. While such an ultimate goal is inspired by the human hallmark of continuous real-time processing of sensory information, several solutions have simplified the learning paradigm by artificially limiting the processed context or dealing with sequences of limited length, given in advance. These solutions were further emphasized by the ubiquity of Transformers, which initially overshadowed the role of Recurrent Neural Nets. However, recurrent networks are currently experiencing a strong recent revival due to the growing popularity of (deep) State-Space models and novel instances of large-context Transformers, which are both based on recurrent computations that aim to go beyond several limits of currently ubiquitous technologies. The fast development of Large Language Models has renewed the interest in efficient solutions to process data over time. This survey provides an in-depth summary of the latest approaches that are based on recurrent models for sequential data processing. A complete taxonomy of recent trends in architectural and algorithmic solutions is reported and discussed, guiding researchers in this appealing research field. The emerging picture suggests that there is room for exploring novel routes, constituted by learning algorithms that depart from the standard Backpropagation Through Time, towards a more realistic scenario where patterns are effectively processed online, leveraging local-forward computations, and opening new directions for research on this topic.

1. Introduction

Human cognition is inherently intertwined with the seamless processing of sequential patterns (Conway & Christiansen, 2001; Elman, 1990; Simon & Kotovsky, 1963). Sequential data are ubiquitous in the context of perception: from the flow of natural language during conversations to the sequence of cues processed in visual perception, and more. Mimicking the human ability to comprehend and learn from sequences has long been an aspiration within the realm of Artificial Intelligence (AI), with the ultimate goal of mirroring the human capacity to retain long-term insights from past experiences while remaining attuned to upcoming information (Lipton et al., 2015; Sodhani et al., 2020). The relevance of processing sequential data is evidenced by the increasing amount of tasks tackled with Deep Learning for scientific and industrial applications, such as conversational AI (Brown et al., 2020), natural language understanding (Wang et al., 2018), video representation learning and processing (Selva et al., 2023), lifelong and continual learning (Ehret et al., 2020; Sodhani et al., 2020), time-series analysis (Bianchi et al., 2017), temporal graphs (Wu et al., 2020), and various other domains (Baccouche et al., 2011; Jurtz et al., 2017).

Early attempts to deal with sequential data date back to the dawn of Machine Learning (Elman, 1990; Hopfield, 1982; McCulloch & Pitts, 1943; Pearlmutter, 1989; Rumelhart et al., 1985), and they focused on the proposal of novel *architectural* and *algorithmic* solutions that departed from popular approaches designed for static data, and were better suited to the context of processing sequences. Indeed, Recurrent Neural Networks (RNNs) go beyond feed-forward models for static data thanks to the way hidden states are structured, serving as a form of memory and being influenced by previous-time self-connections (Elman, 1990; Salehinejad et al., 2017). Given the novelty of such architectural topologies,

* Corresponding author.
E-mail addresses: matteo.tiezzi@iit.it (M. Tiezzi), m.casoni@student.unisi.it (M. Casoni), alessandro.betti@imtlucca.it (A. Betti), marco.gori@unisi.it (M. Gori), stefano.melacci@unisi.it (S. Melacci).

https://doi.org/10.1016/j.neunet.2025.108039
Received 13 June 2024; Received in revised form 4 August 2025; Accepted 25 August 2025
Available online 29 August 2025

several ad-hoc learning rules were proposed (Williams & Peng, 1990; Williams & Zipser, 1989), among which the Backpropagation Through Time (BPTT) (Werbos, 1990) algorithm emerged and became prominent. However, BPTT requires the recursive unfolding of the network over the processed sequences, storing intermediate computations across the whole input sequence, and virtually obtaining a "deep" feed-forward computational graph on which standard back-propagation of errors is applied (Ororbia et al., 2020).

The limits and drawbacks of this solution become evident when the goal is to emulate abilities which are typical of human cognition, such as real-time/online sequence processing. Indeed, the unfolded networks represent very deep pathways to be traversed by error information. As a result, RNNs trained by BPTT suffer from *vanishing gradients* (Bengio et al., 1994; Sodhani et al., 2020) making *credit assignment* much more difficult (Ororbia et al., 2018), i.e., the task of computing the impact on the overall error of the individual units in the network. Early solutions often necessitated compromises to make learning affordable, simplifying the learning paradigm by reducing the context window (e.g., Truncated BPTT (Williams & Peng, 1990)) or assuming the availability of complete sequences in advance. Yet, the aforementioned issues and these simplifications constrained the ability to capture intricate long-term dependencies and correlations that characterize many real-world sequential datasets (Gu et al., 2021a).

However, the ultimate goal of mimicking the human capacity to learn over sequences in real-time has inspired several researchers. Amongst others, Williams and Peng proposed*"an on-line algorithm, designed to be used to train a network while it runs; no manual state resets or segmentations of the training stream are required"* (Williams & Peng, 1990), as well as other well-known variants such as Real Time Recurrent Learning (RTRL) (Williams & Zipser, 1989), which, however, suffer from scalability issues (Javed et al., 2023). Architectural solutions were investigated, for instance through the introduction of gating mechanisms (Long Short Term Memories, LSTMs (Hochreiter & Schmidhuber, 1997)) that partially improve the control over vanishing gradients and the error flow. Remarkably, the original work on gating in LSTMs (Gers et al., 2000) was devised as a remedy to the fact that *"a continual input stream eventually may cause the internal values of cells to grow without bound"*. Still, none of these solutions was capable of overcoming the challenges posed by limited computational resources and the quest for efficiency in sequence processing in RNNs, ultimately resulting in their inability to process sequences longer than a few thousand steps (Li et al., 2018; Voelker et al., 2019). Going beyond instances of RNNs, still in the context of neural networks, the scientific literature includes approaches to long sequence processing that are based on Convolutional Neural Networks (CNNs)—see Hè and Kabic (2023) and references therein. CNNs enable the capability of parallelizing inference/learning over the temporal dimension when the entire sequence is available in advance (Bai et al., 2018). The local nature of the convolution results from the design choice of using filters that span a limited time range around each time instant. However, dealing with small filters hinder s the ability to capture very long-term dependencies, a role that is instead fulfilled by stacking multiple convolutional layers.

The ubiquity of Transformers (Vaswani et al., 2017) in recent years has dominated the sequence processing scenario due to several advantages over other existing architectures. In fact, the training procedure is parallelizable on modern hardware when the full sequence is available (Zhuang et al., 2023). The self-attention mechanism introduces several advantages, such as the ability to handle both local and long-range relations (Tay et al., 2020), while completely avoiding the vanishing gradient issue, thanks to the direct connection of any token pairs in the sequence. However, the *quadratic* complexity characterizing self-attention yields a significant computational cost and memory demand, particularly pronounced when handling long input sequences, which becomes a strong limitation in the case of on-the-edge devices with limited computational resources. These issues have instigated a profusion of research endeavors aimed at improving the scalability attributes of Transformers, often entailing trade-offs with certain traits that underlie their efficiency (Dao et al., 2022; Wang et al., 2020). Several of these approaches are inspired by insights derived from RNNs, that are fast-inference machines that scale *linearly* with the sequence length (Peng et al., 2023).

In parallel to these developments, and during the same years in which Transformers were gaining widespread adoption, an alternative and independent research direction emerged, based upon the usage of State-Space Models (SSMs) for sequence modeling. Inspired by methods developed for continuous-time recurrent neural networks (Gu et al., 2020a; Voelker et al., 2019), this line of work proposed SSMs as a principled way to handle long-range dependencies while preserving efficiency. In particular, a subclass of linear SSMs gained traction, promoting the idea of removing nonlinearities from the state-update rule to enable efficient computations (Gu et al., 2021a). When discretized with ad-hoc integrators, continuos time SSMs were used as additional modules to gated RNNs (Gu et al., 2020a; Voelker et al., 2019). This intuitions inspired a plethora of works aiming at injecting linear SSMs into Deep Architectures (Gu et al., 2021b; Gupta et al., 2022b; Smith et al., 2022), frequently considering diagonal recurrent matrices and further blurring the line between these model classes. More recently, the authors of (Orvieto et al., 2023) proposed a recurrent model that bridges RNNs and the intuitions behind the aforementioned categories of SSMs, referred to as Linear Recurrent Units (LRUs). Basically, they showed that appropriately parametrizing and structuring RNNs (linear and diagonal recurrence, specific initializations, etc.) the advantages of Deep-SSMs can be fully exploited also in standard RNNs. When paired with appropriate gating functions and very structured deep networks, these models can reach state-of-the art results in language modeling (De et al., 2024). While it is useful to talk about Deep-SSMs and RNNs in a separate manner to better emphasize the recent literature on linear models (frequently based on diagonal recurrent matrices and additional neural layers on top of them), we note that RNNs are also instances of state-space models. We summarize in Table 1 the aforementioned ingredients constituting the novel components of current solutions, which we will explore in greater detail in the remainder of the paper. While previous approaches did not consider many of these architectural components–such as linear and element-wise recurrence and gating mechanisms–they have proven fundamental for effectively managing increasingly demanding long-term sequences. When the sequence length is taken to the extreme, i.e., dealing with potentially infinite sequences, the current learning paradigm based upon BPTT must be reconsidered by considering online learning scenarios.

Motivation and Scope In the present era, a confluence of factors, including the rise of large-scale language models (Brown et al., 2020) and the need to optimize their performance during both training and inference, has resulted in a resurgence of interest in recurrent architectures (Orvieto et al., 2023; Peng et al., 2023) and their optimization schemes, marking a new chapter in the narrative of sequence processing, going beyond the *"attention is all you need"* conjecture (Vaswani et al., 2017) (Fig. 1). This survey presents a comprehensive exploration of this recent evolution, covering the latest architectural solutions inspired by recurrent computations, ranging from Transformers with extensive context to the rekindling of interest in linear recurrent networks, driven by multiple variants of deep state-space models. Meanwhile, innovative learning algorithms that challenge the conventional Backpropagation Through Time (BPTT) have emerged. These algorithms represent a significant shift from the traditional methodology, addressing the practical need for online sequence processing (Marschall et al., 2020). They leverage localized and forward-facing strategies to navigate the nuances of real-time pattern recognition (Javed et al., 2023). The primary goal of this survey is to offer an exhaustive taxonomy of contemporary trends in both architectural design and algorithmic innovation for sequence processing with recurrent architectures. By shedding light on the cutting-edge techniques, we aim to provide researchers with a comprehensive guide through the dynamic landscape of sequence processing, guiding them in their pursuit of effective solutions. As the AI community stands on the

Table 1
Features of previous (upper part) and current generation of neural models (bottom part), from the perspective of recurrence. While previous generation did not consider many of those features that usually characterized recurrent models, the current generation (Transformers, state-space models, modern RNNs) share several of such features (linear and element-wise recurrence, gating mechanisms). This trend have proven fundamental when considering extremely long-term dependencies, both in term of performances and scalability.

		Linear Rec.	Element-wise Rec.	Gating	Online learning
Previous Gen	Transformers	–	–	–	–
	RNNs	–	✓	✓	✓
	ConvNets (for sequences)	–	–	–	–
Current Gen	Section 3: Transformers embracing Recurrence	✓	✓	✓	–
	Section 4: Deep State-Space models (SSMs)	✓	✓	✓	–
	Section 5: Enhancing RNNs	✓	✓	✓	–
	Section 6: Learning in RNNs	✓	✓	✓	✓

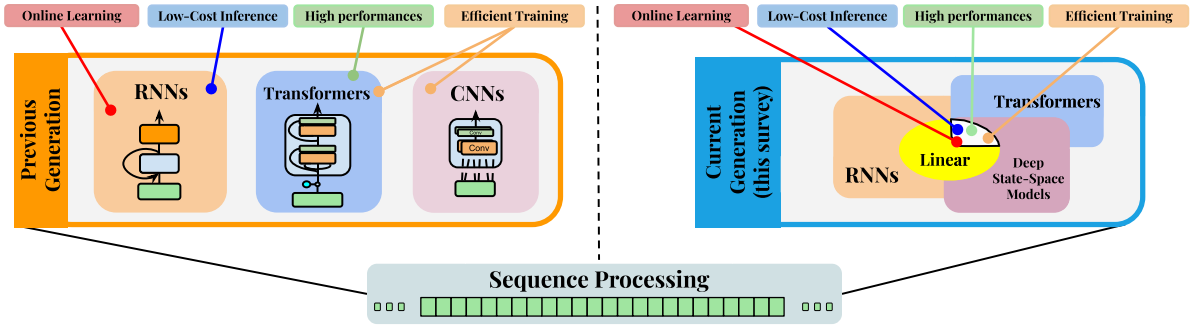

Fig. 1. Left: Since the advent of Transformers (Vaswani et al., 2017), the neural network community has promoted these architectures as alternatives to recurrent (RNNs) and convolutional (CNNs) models for sequence processing, i.e., *"attention is all you need"*, thus they are represented by disjoint boxes in the picture, with the main features characterizing each of them listed at the top. **Right:** In the current generation of neural models, described in this survey, attention has "married" recurrence, generating a significant intersection between the different "worlds". RNNs are state-space models, and in recent years we have observed the emergence of several variants of (deep) State-Space Models, where linearity (yellow set) is one of the main keys behind their success (together with element-wise recurrence and gating functions), and convolutional layers are now parts of various architectures. Interestingly, linearity intersects all the three families of models (RNNs, Transformers, Deep State-Space Models—the latter are almost fully based on linear models). The increasing interest in lifelong-Learning and applications running on limited-computation devices and with privacy constraints, such as instances of edge computing, is shifting the interest toward online learning (previously a prerogative of stateful/RNNs models). The white area is the meeting point of all the important features listed on top, and where there might be more room for novel research activities.

threshold of unprecedented opportunities catalyzed by the advent of large language models, we invite researchers to embark on this journey into the intricate realm of sequence processing. This survey also highlights emerging research directions that lie at the intersection of online learning from sequential data and lifelong learning (Mai et al., 2022). Fig. 1 summarizes the big picture of this paper, emphasizing the"marriage" between RNNs, Transformers, and the evolution of state-space models based on linear units. In a nutshell, stateful models are not only back in the context of learning from sequential data, but they also bring together approaches that were previously considered distinct by the scientific community. New research opportunities may emerge at the intersection with online lifelong learning.

Paper Overview Fig. 2 describes the way this paper is organized. In each section that surveys multiple categories of models, we will further provide a picture to summarize its contents and main methods. Moreover, we include references to previous efforts in surveying sequence processing based on recurrent neural networks. In Table 2, we report the main mathematical symbols used throughout the paper, together with their descriptions and dimensionalities. We start this survey by describing, in Section 2, recurrent models for sequence processing and introducing the notation we will use in the remainder of this manuscript. We will formalize our derivations using a notation that is applied consistently across apparently different models, to facilitate comparisons among them. Section 3 surveys the latest findings regarding the optimization of Transformers for long sequence processing, focusing on methods that aim to improve self-attention mechanisms inspired by recurrent approaches. Section 4 provides a thorough description of the emerging field of deep state-space models. In Section 6, we focus on learning dynamics, describing the recent learning algorithms that challenge the conventional Backpropagation Through Time (BPTT), in pursuit of local and forward-facing methods designed to emulate the human ability to process sequences in an online manner. Then, in Section 7, we

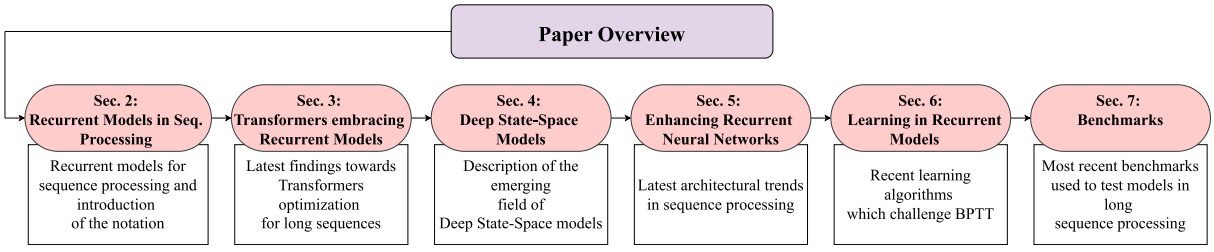

Fig. 2. Overview of the organization of this survey.

Table 2
Summary of the main mathematical symbols used throughout the paper, including RNNs, Transformers, and Deep State-Space Models, along with their description and dimensionality.

Symbol	Description	Dimension
L	Sequence length	–
x	Hidden state	d_s
u	Input vector	d_{in}
y	Output vector	d_y
A	State transition matrix	$d_s \times d_s$
B	Input matrix	$d_s \times d_{\text{in}}$
C	Output matrix	$d_y \times d_s$
D	Feedthrough matrix	$d_y \times d_{\text{in}}$
θ	Learnable parameters	–
σ	Activation function	–
q, k	Query, key vectors	d_k
v	Value vectors	d_y
U	Stacked input (sequence)	$L \times d_{\text{in}}$
Q, K	Stacked query/key (sequence)	$L \times d_k$
V	Stacked value (sequence)	$L \times d_y$
W_q, W_k	Linear query/key projection matrices	$d_k \times d_{\text{in}}$
W_v	Linear value projection matrices	$d_y \times d_{\text{in}}$

survey the most recent benchmarks used to evaluate how well models capture long-term dependencies. Finally, in Section 8, we analyze some open problems and issues that have been recently pointed out in the context of the described works, as well as highlight possible future avenues for research, and we draw our conclusions in Section 9.

2. Recurrent models in sequence processing

In several real-world scenarios, data is organized in a sequential manner, where ordering matters and requires appropriate computational models. This is the case of natural language text, speech, vision, time series, and others. Leveraging the temporal relations and ordering of patterns can help in discovering many additional cues in data (Elman, 1990). Let us describe a sequence of L patterns with the notation

$$(u_1, \dots, u_{t-1}, u_t, u_{t+1}, \dots, u_L), \tag{1}$$

where L is a positive integer that represents the sequence length and $u_t \in \mathbb{R}^{d_{\text{in}}}$, $t = 1, \dots, L$, is the t-th pattern.[1] Note that while t is commonly interpreted as the index of a pattern in the sequence, in many settings (especially in the context of perception or time-series) we also have the use of the time at which the sample was provided or, equivalently, of the length of the time interval that passed between consecutive patterns. We also remark that, in principle, L could be infinite. In this paper, to keep the notation simple, we use t both as the index of a pattern or step and as the time variable, depending on the context.[2]

Stateless Models The most straightforward approach for handling sequences, which have been considered since before recurrent architectures became ubiquitous, is to explicitly inject time by means of a spatial representation (as discussed in Elman (1990), for example). The order of events in a sequence is simply considered to be an additional dimension of the pattern, which can be processed in a single shot. This also allows to design models that process in parallel the information at the different time instants, i.e., without having to wait for computations on past data to finish. There exists several works in the late eighties described by this route (Elman & Zipser, 1988; Hanson & Kegl, 1987), which has been also followed by more recent works based on convolutions (Gu et al., 2021a) and Transformers (Vaswani et al., 2017). The former virtually slides fixed-length filters over the input sequence (Gehring et al., 2017; Smith et al., 2022), the latter leverages positional encodings, in both cases parallelizing computations over time (Dufter et al., 2022). Overall, these models are *stateless*, in the sense that they do not try to build and progressively update an internal representation of the sequence observed so far. As a consequence, they require the whole sequence to be available and/or to reconsider it all if it gets updated, thus not being suited, for example, for continuous online-learning (Gori & Melacci, 2023).

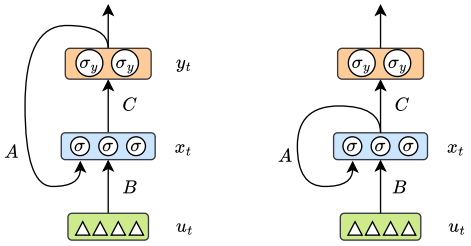

Fig. 3. Jordan (left) and Elman (right) implementations of recurrent models. A, B, C represent linear projections, and input (triangles), hidden (circles-blue boxed), output (orange-boxed) units are emphasized (σ, and σ_y are activation functions).

Recurrent Models An alternative approach consists in focusing on *stateful* models that specifically embrace time in their computational scheme, developing a form of memory that gets updated over time (i.e., the *state*). This is achieved by introducing feedback connections, yielding instances of what are commonly referred as Recurrent Models. Processing a sequence in Recurrent Models consists of processing one pattern at-a-time, following the original order of the data. Internal units are used to model the so-called *state* or *context*, and, due to the feedback connections, they are responsible for integrating information from the current patterns and information from the previous time step, in order to update the state representation. As a result, the state effectively encodes the temporal characteristics of the sequential input observed so far Elman (1990). Several seminal works investigated this very natural direction (Rumelhart et al., 1985; Werbos, 1988). Jordan (1997) introduced a network with one-to-one connections from output units to state units, as shown in Fig. 3-left.

Feedback connections enable the hidden units to access the prior outputs of the network, thereby shaping subsequent behaviors. This property grants the network a form of memory, facilitating the association of static patterns (referred to as "Plans") with serially ordered output patterns (sequences of "Actions"). Elman (1990) proposed the best known instance of a Recurrent Model, where state units interact exclusively with internal nodes of the network, and not with the external world (i.e., state units are fed with the hidden information of the previous step, instead of what comes from the output layer), as shown in Fig. 3-right. There exists an important connection between stateful Recurrent Models and the generic notion of *state-space model*, which has been used in many fields of engineering (Kitagawa, 1998). This notion is common in control theory to model a dynamical system, and it is generic enough to cover a large number of possible implementations, including linear and non-linear systems, whether time-varying or time-invariant. However, in the context of the majority of the literature in Machine Learning, it is rarely mentioned together with Recurrent Models. Koopman theory offers a principled alternative to RNNs for modeling nonlinear dynamical systems by embedding their evolution into a linear function space via the Koopman operator. This perspective enables more stable and interpretable models, as testified by works that incorporate Koopman theory into deep learning through physics-informed autoencoders that outperform traditional RNNs (Azencot et al., 2020). See Section 6 for further discussion on the connections between RNNs and dynamical systems.

[1] This holds both when u_t is considered to be the external input of a computational model or the result of intermediate computations in a multi-layer network.

[2] In the following, the symbol t will be written as a subscript when referring to a discrete-time index, for example u_t, and enclosed in parentheses when representing a continuous-time variable, for example $u(t)$.

State-space models define the temporal evolution of state variables by first-order differential equations or difference equations. The state changes over time as a function of its value at a given instant and the currently provided external input. This definition intrinsically covers the typical feedback connection of Recurrent Models, which can be considered instances of state-space models, as we will formalize in what follows.

Recurrent Neural Networks Elman's architecture (Elman, 1990) was the pillar to the foundations of the widespread notion of Recurrent Neural Network (RNN), summarizing a neural model with a special hidden-layer including lookback connections, that we refer to as RNN layer. Given an input sequence $(u_1, u_2, \dots, u_L)$, an RNN layer with d_s-dimensional hidden state x_t computes a sequence of d_y-dimensional outputs $(y_1, y_2, \dots, y_L)$ through a recurrent model,

$$x_t = \sigma(Ax_{t-1} + Bu_t), \quad y_t = \sigma_y(Cx_t + Du_t), \tag{2}$$

starting from some $x_0 \in \mathbb{R}^{d_s}$, with learnable *state matrix* $A \in \mathbb{R}^{d_s \times d_s}$, *input matrix* $B \in \mathbb{R}^{d_s \times d_{\text{in}}}$, an *output matrix* $C \in \mathbb{R}^{d_y \times d_s}$ and an optional *feed-through matrix* $D \in \mathbb{R}^{d_y \times d_{\text{in}}}$.[3] We denote with σ and σ_y non-linearities on the state and output computation, respectively, often chosen to be the hyperbolic tangent or sigmoid activation. If σ is the identity function, then we say the RNN layer is *linear*. The relation between discrete computation of Eq. (2) and a continuous-time formulation becomes evident once we explicitly introduce the dependence on time and model the variation of the state as follows,

$$\dot{x}(t) = -x(t) + \sigma(Ax(t) + Bu(t)), \; y(t) = \sigma_y(Cx(t) + Du(t)). \tag{3}$$

Moving the term $-x(t)$ to the left-hand side of the first equation yields an evident connection to Eq. (2). This formulation is well-suited to trace another important link with the already introduced notion of *state-space model*. The general form of a state-space model is actually close to the one of Eq. (3), when also including the direct dependence on time t in both the equations, i.e., $\dot{x}(t) = f(x(t), u(t), t)$ and $y(t) = h(x(t), u(t), t)$, being f and h two generic functions. The discrete counterpart of the first equation, when considering time invariant systems, intrinsically yields the classic feedback of Recurrent Models.

Learning in Recurrent Models Recurrent Models are commonly exploited to learn a mapping from the input sequence $(u_1, u_2, \dots, u_L)$ to a target output, being it another temporal sequence $(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_L)$ or a single output vector $\hat{y}_L$. The former can be considered a more general formulation that includes the latter as a degenerate case. At each step t, an instantaneous loss function $\ell(y_t, \hat{y}_t)$ quantifies to what degree the predicted output y_t matches the target output $\hat{y}_t$. Let us collect the model learnable parameters in $\theta := \{A, B, C, D\}$. BPTT (Rumelhart et al., 1985) is the de-facto standard algorithm to learn with Recurrent Models, and works by minimizing the following global empirical risk function over the whole sequence,

$$\mathcal{L}(\theta) = \frac{1}{L} \sum_{t=1}^{L} \ell\big(y_t, \hat{y}_t\big), \tag{4}$$

by gradient descent. Exploiting the chain rule (Werbos, 1990) it can be shown that the gradient for each step k is a sum of products of partial gradients,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} &= \sum_{t=1}^{L} \frac{\partial \ell(y_t, \hat{y}_t)}{\partial \theta} \\ &= \frac{1}{L} \sum_{t=1}^{L} \frac{\partial \ell(y_t, \hat{y}_t)}{\partial y_t} \frac{\partial y_t}{\partial x_t} \sum_{j=1}^{t} \Big(\prod_{s=j}^{t} \frac{\partial x_s}{\partial x_{s-1}} \Big) \frac{\partial x_{j-1}}{\partial \theta}, \end{aligned} \tag{5}$$

where $\prod_{s=j}^{t} \frac{\partial x_s}{\partial x_{s-1}}$ is the term that transports the error back in time from step t to step j. A straightforward way to visualize BPTT is to virtually replicate the model at each time step, generating a deep feed-forward network over the input sequence. The model parameters are virtually "copied" at each time step, that is what is called temporal unfolding of the model. In Fig. 4 we report a sketch to emphasize the way the information propagates over time (black) and the main learning signals from the instantaneous loss function and those going backward over time (red), where for ease of notation we denoted the instantaneous loss $\ell\big(y_t, \hat{y}_t\big)$ with ℓ_t.

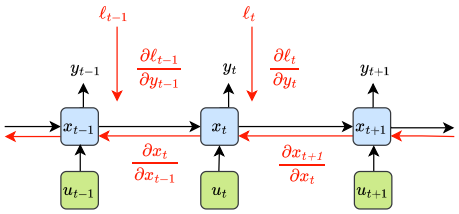

Fig. 4. BPTT. This sketch (Pascanu et al., 2013) emphasizes the way the information propagates (black) and the way the learning signal goes backward in time and down from the instantaneous loss (red). We denote by ℓ_t the loss value at time t.

Issues in BackPropagation Through Time We can rewrite the term $\prod_{s=j}^{t} \frac{\partial x_s}{\partial x_{s-1}}$ of Eq. (5) in the form of a product of Jacobi matrices (Pascanu et al., 2013),

$$\prod_{s=j}^{t} \frac{\partial x_s}{\partial x_{s-1}} = \prod_{s=j}^{t} A' \text{diag}\big(\hat{\sigma}(x_{s-1})\big) \tag{6}$$

where A' is the matrix transpose, $\text{diag}(\cdot)$ converts a vector into a diagonal matrix, and $\hat{\sigma}$ is the derivative of the activation function σ in Eq. (2), which is applied in an element-wise fashion to its input vector. Unless the partial terms $\frac{\partial x_s}{\partial x_{s-1}}$ are close to 1, the product in Eq. (6) could explode or vanish (Hochreiter, 1991). In details, in the simplified case of a linear model (i.e., replacing σ with the identity function) the power iteration method helps in deriving tight boundaries. In particular, a sufficient condition for long term information to vanish is that of having the largest eigenvalue of the recurrent weight matrix A smaller than one, i.e. $\lambda < 1$. Conversely, $\lambda > 1$ is a necessary condition for gradients to explode (see Pascanu et al. (2013) for further details). When gradients undergo vanishing during their backward propagation through time, the critical credit assignment aspect of backpropagation is compromised. In particular, the information about minor state changes in the distant past loses the capacity to influence future states. Conversely, when gradients explode, gradient-based optimization algorithms encounter substantial difficulties in traversing the cost surface. This is primarily due to the assumption that gradient-based optimization relies on that small parameter adjustments yield small changes in the objective function. As the number of time steps considered in the sequence of states increases, the amplifying or diminishing effects associated with the state-to-state transformations at individual time steps can grow exponentially, making it difficult for these models to capture long-range dependencies in the data.

Properties of Recurrent Models The sum-product of L terms in Eq. (5) results in $\mathcal{O}(L^2)$ scaling for each processed sequence. In practice, efficient gradient propagation schemes have been devised to achieve a cost linear in the length of the sequence (Gruslys et al., 2016; Kag & Saligrama, 2021b). In terms of memory consumption, BPTT requires storing all the L intermediate states, resulting in a $\mathcal{O}(L)$ overhead (see also Gruslys et al. (2016) for alternative methods to achieve more memory-efficient BPTT). Since RNN are stateful models, it is not possible to parallelize their inference/training procedure over the time instants to which the components of the sequence belong.[4] Computing the t-th state is conditioned on the current input u_t and the preceding state x_{t-1}, which summarizes all the past knowledge and behaves as an

[3] $D \neq 0$ basically introduces a skip connection (standard in modern architectures) that was not included in the original Elman network (Elman, 1990).

[4] See Section 4 for special cases that can be parallelized.

information bottleneck. In other words, RNNs are *causal* models, since they can only leverage past information available in the input sequence. Indeed, to produce an output y_{t+1}, the model can only access past information up to u_t. A major advantage of this computational scheme is that, during inference, RNNs only require constant computational and storage resources per time step, i.e., $\mathcal{O}(1)$. From the theoretical point of view, RNNs have been proven to be able to simulate Universal Turing Machines (Siegelmann, 2012), i.e., they can combine symbolic and sub-symbolic capabilities by running algorithms on a neural substrate. In this direction, some works introduced the unrealistic assumption of neurons with unbounded precision that equals the number of symbols used in the Turing tape (Siegelmann, 2012; Siegelmann & Sontag, 1992). Chung and Siegelmann (2021) relaxed such assumptions by leveraging a dynamically growing memory module made of neurons of fixed precision. Additionally, RNNs are Universal Approximators (Schäfer & Zimmermann, 2006), i.e., they are able to approximate any open dynamical system with an arbitrary accuracy (see Li et al. (2022c) and references therein for further Universal Approximation results for both discrete and continuous RNNs). We further mention several studies on the way recurrent models (with trained and untrained state transition function) handle memorization, such as those that fall in the context of Echo State Networks (Gallicchio & Micheli, 2017). The reader can refer to Section 6 for further details on the literature on this topic.

Other Surveys on Recurrent Models. Due to the large popularity of RNNs, there are several books (Graves & Graves, 2012), surveys, and reviews that describe their properties and architectural advances. Detailed overviews were included in recent surveys (Lipton et al., 2015; Salehinejad et al., 2017), covering models up to 2015–2017, respectively. An in-depth analysis of gating-based architectures has been given in Yu et al. (2019). Bianchi et al. (2017) provide a general overview on RNNs and learning algorithms before focusing on short-term load forecasting. The BPTT algorithm and its biological plausibility are investigated in Lillicrap and Santoro (2019). An important branch of research looks towards the discovery of online learning algorithms in the context of RNNs, and Marschall et al. (2020) proposes an interesting framework that summarizes recent findings. There are also surveys that focus on the application of RNNs in confined areas, such as continual (Cossu et al., 2021; Ehret et al., 2020) and online learning (Parisi & Lomonaco, 2020).

Recent Findings The usage of Recurrent Models for sequence processing became less popular since 2017, due to the already established popularity of Convolutional models (Bai et al., 2018) and, more importantly, due to the growing ubiquity of Transformer-based solutions (Vaswani et al., 2017), driven by the ease of parallelizing training over the time instants on which the input sequences are defined. However,processing long sequences is hard and expensive with such architectures, fostering the need for novel paths to achieve efficient architectures (Tay et al., 2023). As a result, the scientific literature has recently experienced a resurgence of interest in RNNs (Katharopoulos et al., 2020; Orvieto et al., 2023). Transformers' quadratic complexity with respect to the sequence length has pushed towards the discovery of novel methods to optimize inference (Lin et al., 2022; Tay et al., 2023), and many solutions based on stateful recurrent computations have been introduced (Katharopoulos et al., 2020; Peng et al., 2023; Poli et al., 2023; Sun et al., 2023a). At the same time, the need to preserve extremely long-term dependencies on sequences has led to the adoption of the so-called (deep) State-Space Models (SSMs) (Gu et al., 2020a; Smith et al., 2022). This research field originated from the existing knowledge on RNNs (Voelker et al., 2019), but then it took its own direction. It has been very recently formally reconnected to RNNs by a careful architectural design (De et al., 2024; Orvieto et al., 2023). This is not surprising, since, as anticipated, RNNs are indeed instances of SSMs. In this section, we established the foundational dichotomy between stateless sequence processors and stateful recurrent models, formalised the Elman-type RNN, and analysed BPTT together with its vanishing/exploding gradient pathology. The discussion highlighted RNNs' theoretical universality, their $\mathcal{O}(1)$ inference cost, and their recent resurgence prompted by the scaling limits of Transformers. These insights set the stage for hybrid approaches that attempt to combine recurrent efficiency with modern attention mechanisms.

The classical RNN framework outlined above clarifies both its expressive power and its well-known training bottlenecks (vanishing/exploding gradients and limited parallelism). These limitations have recently motivated a wave of research that injects recurrence into the dominant Transformer family, seeking the best of both worlds–parallel training and explicit long-range memory. The next section surveys this convergence and shows how "attention meets recurrence".

3. Transformers embracing recurrent models

Transformers (Vaswani et al., 2017) emerged as a disruptive alternative to Recurrent Models, promoting the idea of going beyond stateful models that process the sequence one element at a time, motivated by the *"attention is all you need"* motto. Transformers are able to handle the elements of an input sequence (Tay et al., 2023) in parallel and to capture both local and long-range dependencies. These properties are due to their self-attention-based computational scheme, which compares all the possible pairs of input patterns constituting the sequence. In the following, we present a brief analysis aimed at summarizing the properties of self-attention and subsequently underline its drawbacks. Finally, we will delve into the latest findings on self-attention variants and alternatives that, perhaps unexpectedly, can be seen through the lens of Recurrent Models. Fig. 5 showcases the organization of this section.

Transformers and Self-Attention Given the input sequence $(u_1, u_2, \dots, u_L)$ with $u_t \in \mathbb{R}^{d_{\text{in}}}$, $t = 1, \dots, L$, Transformers implement a sequence-to-sequence function $\texttt{tr} : \mathbb{R}^{L \times d_{\text{in}}} \to \mathbb{R}^{L \times d_y}$, that yields $(y_1, y_2, \dots, y_L)$. The whole function tr is based on a self-attention procedure, followed by a feed-forward network, and it is commonly applied multiple times in a multi-layer fashion.[5] Self-attention acts across the temporal dimension of the input sequence, evaluating pairwise interactions between the input components, regardless of their positions in the sequence,[6] and it is evaluated (in parallel) on each element of the input sequence. When evaluated on a generic u_t, it returns a sum over $(u_1, u_2, \dots, u_L)$, weighed by scores that depend on the similarities between u_t and the L elements of the input sequence. Formally,

$$y_t = \sum_{i=1}^{L} \frac{\text{sim}(q_t, k_i)}{\sum_{j=1}^{L} \text{sim}(q_t, k_j)} v_i, \quad t = 1, \dots, L, \tag{7}$$

where $q_\cdot$, $k_\cdot$, $v_\cdot$ are the so-called *queries*, *keys* and *values*, respectively, computed by projecting (three linear projections) the elements of the input sequence using three trainable matrices $W_q, W_k \in \mathbb{R}^{d_k \times d_{\text{in}}}$ and $W_v \in \mathbb{R}^{d_y \times d_{\text{in}}}$, where d_k is the keys and query size while d_y is the self-attention output size. We have $q_z = W_q u_z$, $k_z = W_k u_z$ and $v_z = W_v u_z$, for $z = 1, \dots, L$. The similarity function $\text{sim}(q_\cdot, k_\cdot)$ returns a positive score that increases with the similarity between its arguments.

The canonical choice for the self-attention function is the *softmax dot-product* (multiplicative) attention, where the similarity score is computed as the exponential of the dot product between a query and a key, $\text{sim}(q_\cdot, k_\cdot) = \exp\left(q_\cdot' k_\cdot / \sqrt{d_k}\right)$, where $'$ is the transpose operator and all mono-dimensional arrays are assumed to be column vectors (here and in the rest of the paper).

[5] We purposely avoid describing these components and others, such as skip connections, layer normalization, and multi-head attention. We also do not mention the distinction between Transformer encoders and decoders, since they do not introduce useful information for the point we make. Please refer to Tay et al. (2023) and references therein for detailed descriptions of the Transformer architecture.

[6] Positional embeddings are commonly exploited to make Transformers position-dependent.

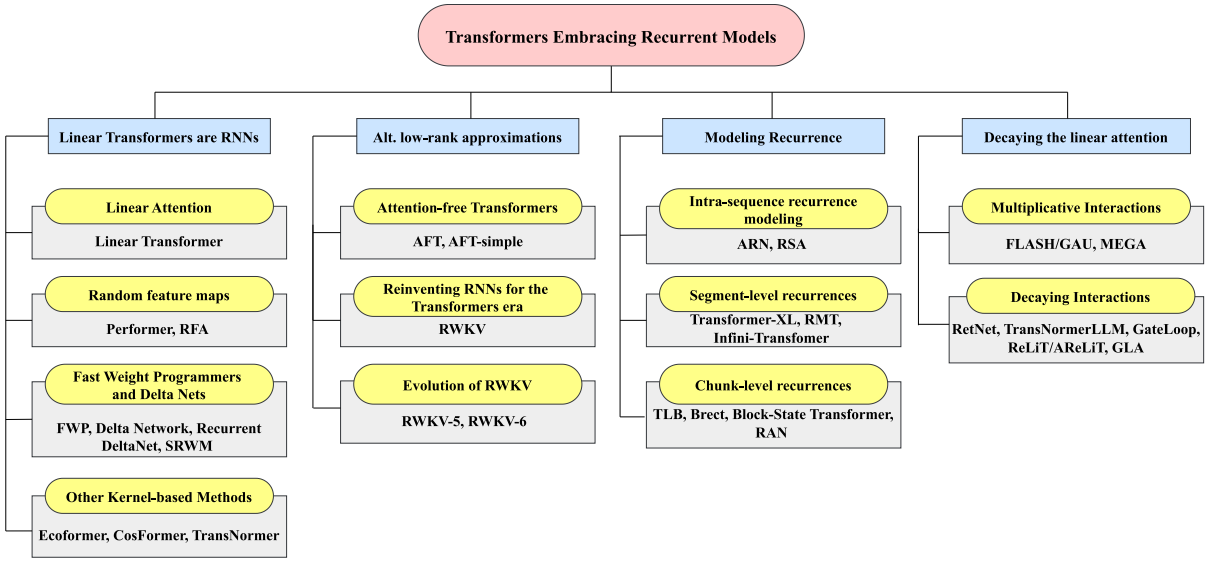

Fig. 5. Conceptual overview of the organization of Section 3,which categorizes Transformer-based models according to architectural and functional aspects. We report the names of the subsections (blue boxes) and the covered categories of models (yellow). The names of some representative methods are also indicated for each category (gray).

Before going into further details, we introduce the matrix notation that we will use throughout this section to simplify the descriptions. Matrix U is composed of L rows, each storing the transpose of u_t, for $t = 1, \dots, L$. Similarly, for all the already introduced symbols that are associated with the L elements of the input sequence, their uppercase versions (with no subscripts) indicate matrices with L rows (e.g., in the case of y_t, q_t, k_t, v_t, $t = 1, \dots, L$, we have matrices Y, Q, K, V, respectively). We can rewrite Eq. ((7) in matrix notation as follows,

$$Q = UW'_q, \quad K = UW'_k, \quad V = UW'_v,$$
$$Y = \mathrm{softmax}\left(\frac{QK'}{\sqrt{d_k}}\right)V \tag{8}$$

where $\mathrm{softmax}(A)$ is the softmax function that operates on each row of A. The above formulation is usually referred to as the *parallel form* of attention, since it computes the Transformer outputs for all the time instants of the sequence (i.e., the full matrix Y) in parallel given the full input matrix U, meaning that all the sequence must be available in advance. This is one of the main advantages of Transformers: during training, which is usually performed using the full input sequence, computations can be efficiently parallelized over the sequence dimension–hence the term *parallel training*.

Causal Attention Unlike the aforementioned scenario where the full sequence is available in advance, there are settings where future information in the sequence cannot be accessed, thus requiring the implementation of a form of *causal attention*. For instance, Transformers can work in an *autoregressive* setting, where the input sequence is progressively populated in an iterative manner (adding an element predicted by the Transformer itself at each time step, e.g., during inference while decoding). The standard self-attention of Eq. (7) is not causal, given that future positions $j > i$ can influence the current one. However, autoregressive Transformers can be obtained by causal masking, i.e., masking the attention computation such that the i-th position can only be influenced by preceding ones. In this causal setting, the parallel form of Eq. (8) can be rewritten as $Y = \mathrm{softmax}\left(\frac{QK'}{\sqrt{d_k}} \odot M\right)V$, where $M \in \mathbb{R}^{L\times L}$ is a mask that prevents the model from attending to future tokens, i.e., $M_{ij} = 1$ if $i \geq j$ and $M_{ij} = -\infty$ if $i < j$. During inference, causal Transformers can be interpreted as exploiting a *recurrent form*, which is obtained by considering only the time steps up to the current one in Eq. (7), i.e., $y_t = \sum_{i=1}^{t} \frac{\mathrm{sim}(q_t,k_i)}{\sum_{j=1}^{t}\mathrm{sim}(q_t,k_j)} v_i$, $t = 1, \dots, L$. Note, that, through this causal mechanism, attention is performed over sets of keys and values that grow over time: to compute y_{t-1}, the sequences $(k_i)_{i=1}^{t-1}$ and $(v_i)_{i=1}^{t-1}$ are needed; to compute y_t, the additional key and value k_t and v_t must be considered in the computation as well, thus expanding the aforementioned sequences of keys and values over time also referred to as the KV-cache (Pope et al., 2023).

Computational Cost and Scaling Issues As stated above, one of the key contributions of the Transformer architecture is that inference can be parallelized over the temporal dimension (Tay et al., 2023), if the full sequence is available in advance, thus overcoming the intrinsic limitations of stateful Recurrent Models. In fact, all the matrix-by-matrix products in Eq. (8) can be computed as L independent vector-by-matrix products. Thus, there is evident room for parallel implementations, given the full input U. Despite the significant benefits brought by parallelization, Transformers are characterized by serious computational and memory costs. In fact, unlike Recurrent Models that keep track of the prior context via an explicit state with fixed size, Transformer self-attention has a "global" receptive field that directly attends the whole sequence at inference. As a result, the main bottleneck is constituted by the self-attention, which scales quadratically with the sequence length, $\mathcal{O}(L^2)$. Indeed, from the definition in Eq. (8): computing $QK'/\sqrt{d_k}$ takes $\mathcal{O}(L^2 d_k)$; the softmax function involves elementwise exponentiation (and sum) of its matrix argument, taking $\mathcal{O}(L^2)$ time, and division of each element by the corresponding row sum, which takes $\mathcal{O}(L^2)$; multiplication of $\mathrm{softmax}(\cdot)$ and V takes $\mathcal{O}\big(L^2 \max(d_k, d_y)\big)$ time (Keles et al., 2023); overall, inference has $\mathcal{O}(L^2)$ complexity in the sequence length. In the aforementioned *autoregressive* setting, where the input sequence is progressively populated and keys and values grow over time in the KV-cache, the computational cost of autoregressive inference is $\mathcal{O}(L_t)$, being L_t the length of the accumulated sequence at time t. All the aforementioned complexities make it harder to apply Transformers when scaling up the input sequence length. This issue has forced practitioners to artificially limit the sequence size, considering a custom context window length, thus hindering the model's capability of capturing long-term dependencies. Investigations on such limitations have spurred a number of efforts aimed at improving Transformers' scalability (Tay et al., 2023), with the final goal of retaining their performance, parallel computations, and merging them with efficient inference com-

Table 3
Complexities for sequences of length L, comparing representatives of the categories of Transformers discussed in Section 3, vanilla Transformers and Recurrent Models (Section 2). In order to bridge the discussion in the main text of paper, here we assumed $d_k = d_{\text{in}}$, for the sake of simplicity. The symbol T denotes the segment length in the case of Transformer-XL.

	Model	Recurrent	Time	Space
Section 2	Recurrent Net	Yes	$\mathcal{O}(Ld_{\text{in}}{}^2)$	$\mathcal{O}(Ld_{\text{in}})$
Section 3	Transformer	No	$\mathcal{O}(L^2 d_{\text{in}})$	$\mathcal{O}(L^2 + Ld_{\text{in}})$
Section 3.1	Linear Trans.	Yes	$\mathcal{O}(Ld_{\text{in}}{}^2)$	$\mathcal{O}(Ld_{\text{in}} + d_{\text{in}}{}^2)$
Section 3.2	RWKV	Yes	$\mathcal{O}(Ld_{\text{in}})$	$\mathcal{O}(d_{\text{in}})$
Section 3.3	Transformer-XL	Yes	$\mathcal{O}(T^2 d_{\text{in}})$	$\mathcal{O}(T^2 + Tm)$
Section 3.4	RetNet	Yes	$\mathcal{O}(Ld_{\text{in}}{}^2)$	$\mathcal{O}(d_{\text{in}}{}^2)$

plexity. Among efforts aimed at optimizing the attention mechanism (Beltagy et al., 2020; Choromanski et al., 2020a; Dao et al., 2022; Kitaev et al., 2020) or completely replacing it Zhai et al. (2021), many recent works have been inspired by the advantages and peculiarities of Recurrent Models. We report an overview of the complexities of different variants of recent Transformer models in Table 3, which will be discussed in the following.

3.1. Linear transformers are RNNs: attention through the lens of kernels

Reaching the ambitious goal of reducing the cost of autoregressive inference from $\mathcal{O}(L_t)$ to $\mathcal{O}(1)$ while attempting to retain the performances of vanilla Transformers and still enabling parallel computations (with crucial impact in reducing training times on large datasets) is extremely challenging. In Fig. 1, the three aforementioned properties are depicted at the bottom, with blue, pink, and yellow boxes, respectively. There exist multiple categories of models that were recently proposed, sharing the intuition of introducing linearity in the computations (notice that this term is used in a two-fold manner: the attention process does not exploit non-linear functions, and the goal is to gain linear complexity), sometimes going back to the theory of kernels, to recover stateful models of attention (Fig. 5, first column).

Linear Attention The so-called Linear Transformer (Katharopoulos et al., 2020) achieves a linear complexity on the self-attention operation introducing a similarity function $\text{sim}(\cdot)$ in Eq. (7) described by a kernel function $\mathcal{K}$ (Hofmann et al., 2008; Tsai et al., 2019), $\text{sim}(q, k) := \mathcal{K}(q, k) = \phi(q)'\phi(k)$, where $\phi : \mathbb{R}^{d_k} \mapsto \mathbb{R}^{d_{\text{ker}}}$ is a non-linear feature map and the kernel codomain should be positive in order to define proper attention weights, i.e., $\mathcal{K} : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \mapsto \mathbb{R}^+$. Therefore, leveraging the associative property of matrix multiplication, Eq. (7) can be rewritten as

$$
\begin{aligned}
S_L &= \sum_{j=1}^{L} v_j \otimes \phi(k_j), \\
z_L &= \sum_{j=1}^{L} \phi(k_j), \\
y_t &= \sum_{i=1}^{L} \frac{v_i \phi(k_i)' \phi(q_t)}{\sum_{j=1}^{L} \phi(k_j)' \phi(q_t)} = \frac{S_L \phi(q_t)}{z'_L \phi(q_t)},
\end{aligned}
\tag{9}
$$

where $\otimes$ denotes the outer product between vectors, and S_L is a matrix. This is what is commonly referred to as *cross-attention* or *encoder self-attention* in the Transformers literature, where the whole sequence is available in advance and computing y_t requires the aggregated term S_L. However, the two terms $S_L \in \mathbb{R}^{d_k \times d_{\text{ker}}}$ and $z_L \in \mathbb{R}^{d_{\text{ker}}}$ are computed once and reused for every query position $t \in \{1, \dots, L\}$. Hence, Linear Transformers reduce the space and time complexity requirements to $\mathcal{O}(L)$, as depicted in Fig. 6 (left). This is evident when we express the numerator of the previous equation exploiting the *parallel form*, i.e. $\phi(Q)S_L := \phi(Q)\big(V'\phi(K)\big)$, where the feature map ϕ is applied rowwise to the matrices Q and K. If, without any loss of generality, we consider keys, queries, and values of size d_k and a cost to compute ϕ of $\mathcal{O}(d_{\text{ker}})$, then the overall run-time complexity of Linear Transformer is $\mathcal{O}(Ld_{\text{ker}}d_k)$. The role of the kernel function ϕ has been investigated in several subsequent works (Choromanski et al., 2020b; Peng et al., 2021; Schlag et al., 2021). Linear Transformer (Katharopoulos et al., 2020) select $\phi(x) = \text{elu}(x) + 1$, where $\text{elu}(\cdot)$ denotes the exponential linear unit (Clevert et al., 2015), a kernel that preserves the dimension of the input key vector ($d_{\text{ker}} = d_k$), leading to a globally linear complexity of the model with respect to the sequence length, i.e., $\mathcal{O}(Ld_k{}^2)$, implying less computation (in terms of number of operations) in long sequences.

Causal Linear Attention Interesting properties emerge from the kernel-based formulation when moving to the *autoregressive* setting. As previously anticipated, this intrinsically requires to implement a form of *causal attention*, since future information cannot be accessed. At a generic time t, we need to have access to S_t and z_t (i.e., they are aggregated up to what has been seen so far), and Linear Transformers can be seen as stateful models that, at each time step, update an internal state $X_t := (S_t, z_t)$, composed by a recurrent state matrix S_t and a recurrent normalization vector z_t, which are updated iteratively with what is referred to as *additive* interaction,

$$S_t = S_{t-1} + v_t \otimes \phi(k_t), \tag{10}$$

$$z_t = z_{t-1} + \phi(k_t), \tag{11}$$

$$y_t = \frac{S_t \phi(q_t)}{z'_t \phi(q_t)}, \tag{12}$$

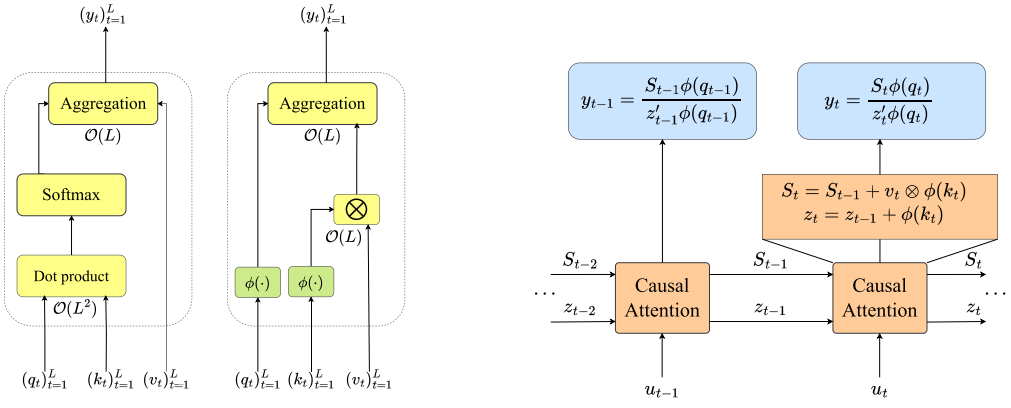

Fig. 6. Left: sketch which highlights the main operations and their complexity in classic softmax attention (left) and in kernel-based linear attention methods (right). Right: how linear attention evolves over time, showing its stateful nature.

with some initial condition S_0, z_0 (usually set to zero). Hence, in an autoregressive setting, y_t can be computed in a recurrent fashion, multiplying the current query with the S_t portion of the state, and normalizing it with its z_t portion. This view challenges the traditional distinction between RNNs as "automata-like" constant-size stateful models and Transformers as not being so. Indeed, autoregressive Transformers can be equivalently expressed as an RNN-like sequence processor with 2-dimensional constant-size states that are updated by sum, as depicted in Fig. 6 (right). Noticeably, this *recurrent* form allows to compress history into the matrix-valued hidden state, thus eliminating the need to maintain and attend over the KV cache during inference.

We remark that, when considering the *parallel* form of causal linear attention, its complexity is still quadratic in L. Indeed, due to the presence of the causal masking matrix M, it is not possible to exploit the associative property of matrix multiplication to reduce the parallel form complexity from quadratic to linear. Noticeably, Linear Transformers combine the best of standard attention and recurrence: at training time (i.e., in non-causal tasks where the full sequence is available in advance), computations can be parallelized and take full advantage of GPUs or other accelerators by leveraging the *parallel form*. When it comes to inference (e.g., sequential decoding), the cost per time and memory for one prediction is constant, and the internal state $X_t = (S_t, z_t)$ can be updated at every time step like in RNNs, thanks to their *recurrent form*. Tasks where vanilla quadratic-time softmax attention cannot be fully parallelized across time steps can be effectively accelerated, such as in autoregressive decoding, both in the conditional (e.g., machine translation) and unconditional (e.g., sampling from a language model) cases (Peng et al., 2021). Additionally, it has been proposed to reduce memory transfer costs by avoiding materializing states in GPU slow HBM memory (see Section 8.2 for further details).

Random Feature Maps Despite the effective computational advantages, Linear Transformers underperform compared to vanilla self-attention, and the main cause has been attributed to the choice of the kernel function (Lin et al., 2022; Peng et al., 2021; Schlag et al., 2021). More recent works (Choromanski et al., 2020a,b; Peng et al., 2021; Schlag et al., 2021) leverage random feature maps to achieve unbiased estimations of shift-invariant kernels, such as the Gaussian one, in order to approximate effects obtained by using the softmax function in vanilla Transformers. Performer (Choromanski et al., 2020a,b) uses random feature maps defined by custom functions $f_1, \cdots, f_l : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$. Formally, given an input vector $k_t \in \mathbb{R}^{d_{\text{k}}}$ (it could be a key or a query),

$$\phi(x) = \frac{h(x)}{\sqrt{m}}[f_i(\omega_1' x), \ldots, f_i(\omega_m' x)]_{i=1}^{l},$$

where the notation $[a_i]_{i=1}^{l}$ indicates a vector obtained by concatenating all the l different a_i's, $\omega_1, \cdots, \omega_m$ are random vectors independently drawn from a distribution $D \in \mathcal{P}(\mathbb{R}^{d_{\text{in}}})$, i.e., $\omega_1, \cdots, \omega_m \overset{\text{iid}}{\sim} D$. Choromanski et al. (2020b) tested trigonometric functions with $h(x) = \exp(\|x\|^2/2), l = 2, f_1 = \sin, f_2 = \cos$, inspired by the random Fourier feature map (Rahimi & Recht, 2007), that has been proved to be successful in speeding up kernel methods (Oliva et al., 2015) and to approximate softmax (Rawat et al., 2019). Given that relying on trigonometric functions does not guarantee non-negative attention scores and thus could lead to unstable behaviors, follow-up works (Choromanski et al., 2020b) proposed positive random feature maps, such as those based on $h(x) = \exp(-\|x\|^2/2), l = 1, f_1 = \exp$, guaranteeing unbiased and non-negative approximation of dot-product attention.

A concurrent work, Random Feature Attention (RFA) (Peng et al., 2021), leverages similar intuitions, building a feature map with $h(x)$ set to 1, with queries and keys ℓ_2-normalized in advance (see Peng et al. (2021) for more details), which shows benefits with respect to the kernels based on the elu. Additionally, RFA includes a learnable gating mechanism inspired by LSTMs (Hochreiter & Schmidhuber, 1997) to explicitly model the notion of *recency bias* and locality, which are not considered in vanilla self-attention. Eqs. (10) and (11) are augmented by means of a gating function returning a score g_t in $(0, 1)$, becoming

$$g_t = \sigma(w_g' u_t),$$
$$S_t = g_t\, S_{t-1} + (1 - g_t)\, v_t \otimes \phi(k_t),$$
$$z_t = g_t\, z_{t-1} + (1 - g_t)\, \phi(k_t),$$

where w_g is a vector of learnable parameters (there could also a bias term), and σ is a sigmoid activation. The multiplicative interaction between the learned scalar gates $0 < g_t < 1$ and the hidden state $X_t = (S_t, z_t)$ exponentially decays past information, favoring more recent contexts. In addition to using random feature maps to approximate standard dot product attention, Peng et al. (2021) and Choromanski et al. (2020b) explore alternative routes, approximating order-1 arc-cosine kernel. In this case with $h(x) = 1, l = 1, f_1 = \text{ReLU}$. This feature map has been show to be effective in various tasks including machine translation and protein sequence modeling.

Fast Weight Programmers and Delta Nets Schlag et al. (2021) showed the formal equivalence between kernel-based Linear Transformers and the seminal works on Fast Weight Programmers (FWPs) (Schmidhuber, 1992, AI Blog, 2021). It turns out that the kernel-based causal perspective of linear attention (Eqs. (10)–(12) and Fig. 6-right) is a FWP with the additional recurrent normalization factor z_t. The intuition behind the notion of *fast weights* is to introduce novel dependencies on the weights of the model. A two-network system is proposed, composed of a slow net with slowly-changing weights, which continually reprograms another net with weights that change in a fast manner, making them dependent on the spatio-temporal context of a given input stream. Notably, this finding suggests that the recurrent state matrix S_t ($\approx W^{(t)}$ in FWP) can be seen as a key-value associative "memory matrix" which gets reprogrammed. (*i*) The "write" operation is obtained by aggregating the results of the outer products of (ϕ-mapped) keys and values in Eq. (10), also referred to as *associations*. (*ii*) The "retrieve" operation consists of multiplying the memory matrix by the (ϕ-mapped) query (see Eq. (12)). Schlag et al. (2021) argue that endlessly adding new associations to a memory of finite size (Eq. (10)) will inevitably reach a capacity limit. To prevent associations from interfering with each other upon retrieval, keys must be orthogonal, i.e., the feature map size must be large enough to avoid working in an overcapacity regime.[7] This analysis showcases the limits and sub-optimality of random feature maps, characterized by a $2m$-sized capacity which would require m to go to infinity to yield robust approximations. Interestingly, Schlag et al. introduce a deterministic parameter-free feature map which fosters orthogonality in feature space. Specifically, the feature map they propose is such that $d_{\text{ker}} = 2\nu d_{\text{k}}$, and it is defined as

$$\phi(x) = \left[\text{ReLU}([x, -x])_i \text{ReLU}([x, -x])_{i+\nu}\right]_{i=1}^{2d_{\text{k}}},$$

with a capacity controlling parameter $\nu \in \{1, \ldots, 2d_{\text{k}} - 1\}$. Similar intuitions led to the enlargement of the associative memory capacity in a write-and-remove fashion. Differently from RFA, associations in the memory are updated while keeping intact other unrelated ones. Specifically, given a new input key-value pair (k_t, v_t), the model first attempts a retrieve operation using k_t, in order to get back an estimated value vector $\bar{v}_t$ accordingly to the currently available memory S_{t-1}, i.e., $\bar{v}_t = S_{t-1}\phi(k_t)$. It then writes to the memory matrix the difference between the real v_t and the estimated $\bar{v}_t$, modulated by a gating function g_t. This update mechanism is an error-correcting delta rule

[7] With keys embedded in a d_{ker}-space, there cannot be more than d_{ker} orthogonal vectors. That is, storing more than d_{ker} associations will result in retrieval issues. Linear attention based on the elu function preserves the dimension of the input key vector ($d_{\text{ker}} = d_{\text{k}}$) without modifying the memory capacity, thus, when $L > d_{\text{ker}}$, the model might be in such an overcapacity regime.

(Widrow et al., 1960) with a dynamic learning rate g_t,

$$\begin{aligned} g_t &= \sigma(w_g' u_t), \\ S_t &= S_{t-1} + g_t(v_t - S_{t-1}\phi(k_t)) \otimes \phi(k_t), \\ y_t &= S_t \phi(q_t), \end{aligned} \tag{13}$$

hence the model is referred to as Delta Network. Notice that the recurrent normalization factor z_t leveraged in Eq. (12) is not there anymore, since the authors propose to apply a sum-normalization before updating the memory matrix (i.e., they normalize $\phi(q_t), \phi(k_t)$ by the sum of their components).

Beyond Delta Nets Follow-up works extended Delta Networks by adding recurrent connections that feedback the previous output y_{t-1} (actually, $\tanh(y_{t-1})$), resulting in Recurrent Delta Networks (Irie et al., 2021). Such recurrent connections are exploited when computing the current key k_t, query q_t, value v_t, and g_t of Delta Networks, i.e.,

$$\begin{aligned} k_t &= W_k u_t + R_k \tanh(y_{t-1}), \quad \text{(same for } q_t \text{ and } v_t\text{)} \\ g_t &= w_g' u_t + r_q' \tanh(y_{t-1}), \\ y_t &= S_t \phi(q_t) + R_y \tanh(y_{t-1}), \end{aligned}$$

where the $R_\cdot$'s and r_q are new projection matrices and vector, respectively. Inspired by Self-Referential Weight Matrix (SRWM) (Schmidhuber, 1993), Irie et al. (2022b) proposed a different approach, which can be considered an extension of previous works on FWPs. Compared to Delta Networks of Eq. (13), SRWM is based on a computational scheme in which the output y_t is directly computed by projecting the (ϕ-mapped) input, $y_t = W_y \phi(u_t)$, being W_y a learnable projection matrix. Such a new projection matrix and the already introduced W_q, W_k, w_g, are the outcome of a programming scheme which consists in a recurrent process, i.e., the one that in Eq. (13) is used to update S. Once we replace S by another matrix $\tilde{S} = [W_y, W_q, W_k, w_g]$, and by appropriately choosing the number of components in v_t, SRWM computes the value vector as a function of $\tilde{S}$,

$$\begin{aligned} [y_t, q_t, k_t, g_t] &= \tilde{S}_t \phi(u_t), \\ v_t &= \tilde{S}_t \phi(q_t), \\ \tilde{S}_t &= \tilde{S}_{t-1} + g_t(v_t - \tilde{S}_{t-1}\phi(k_t)) \otimes \phi(k_t), \end{aligned}$$

where the first equation summarizes the linear projections of the (ϕ-mapped) input by W_y, W_q, W_k, w_g in a compact manner. A very recent work (Irie et al., 2023a) investigates the computational power of Transformers with linear attention (Katharopoulos et al., 2020) and Delta Networks (Schlag et al., 2021), showing that the just introduced recurrent (Irie et al., 2021) and self-referential extensions of the latter (Irie et al., 2022b) provides improvements over Linear Transfomers, e.g., allowing for generalisation on the parity problem. Mao (Mao, 2022) proposes a data-dependent decay-based update, exploiting gated rules (Peng et al., 2021). A low-rank decay matrix G_t with values in $(0, 1)$ is factorized by two gating vector-functions, and it is used to decay the state S_t,

$$\begin{aligned} G_t &= \sigma(W_f u_t)\sigma(W_z u_t)', \\ S_t &= G_t \odot S_{t-1} + v_t \otimes k_t, \end{aligned} \tag{14}$$

where $\odot$ is the element-wise product and $W_z \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$, $W_f \in \mathbb{R}^{d_{\text{k}} \times d_{\text{in}}}$ are newly added learnable parameters. It clearly differs from the delta rule in Delta Networks (Schlag et al., 2021), especially due to the introduction of a finer-grained element-wise operation. Moreover, there is no ϕ function in Eq. (14) since this model virtually exploits a linear feature map $\phi(x) = W_\phi x$, that can be subsumed into the query-key projections W_k, W_q, inspired by Kasai et al. (2021). The hidden states $S \in \mathbb{R}^{L \times d_{\text{k}} \times d_{\text{v}}}$ (which are required for gradient computation during the backward pass) must be stored due to the addition of G_t in Eq. (14), which makes the computation non-reversible (MacKay et al., 2018) and the I/O memory management challenging.

Other Kernel-based Linear Methods There exist other works that we can connect to the principles behind linear attention and recurrence. Ecoformer (Liu et al., 2022b) exploits an energy-saving kernel-based hashing (RBF kernel) to map the queries and keys onto low-dimensional binary codes in Hamming space. Kernelized hash functions are learned in a self-supervised manner, driven by the Hamming affinity of the attention scores. CosFormer (Qin et al., 2022b) uses $\phi(x) = \text{ReLU}(x)$ ensuring non-negativity in the attention weights, and a cosine-based re-weighting mechanism to enforce the locality bias in the original softmax attention. The authors of TransNormer (Qin et al., 2022a) identify issues in kernel-based Linear Transformer, both due to *unbounded gradients* and to *attention dilution*, i.e., the sparse distribution of attention scores in long sequences. They propose to solve them by normalizing the attention computation and leveraging diagonal attention confined to neighbouring tokens in early layers. In particular, TransNormer leverages a linear kernel, replacing the role of the z_t normalization factor of Eq. (12) by Layer Normalization (Ba et al., 2016b) on the attention scores (NormAttention). This results in the most compact form of the update equations of Linear Transformers,

$$\begin{aligned} S_t &= S_{t-1} + v_t \otimes k_t, \\ y_t &= S_t q_t, \end{aligned} \tag{15}$$

from which it is even easier to trace connections to linear RNNs equipped with "matrix-valued" hidden state S_t, updated by the outer-product $v_t \otimes k_t$, as already pointed out by several works (Irie et al., 2021; Mao, 2022; Schlag et al., 2021; Schmidhuber, 1992). We mention that the idea of proposing an RNN-oriented view of matrix-valued state Transformers has been also recently remarked by Oren et al. (2024), introducing Multi-state RNNs. In such a view, the state can be considered the number of input tokens in the sequence, which is basically related to the time-increasing KV-cache.

To sum up, all the papers described up to this point highlight the rich and convenient connections between Transformer models and RNNs, when considering specific architectural designs. In this respect, linear Transformers can be considered as linear RNNs acting in a matrix-valued state space, where states are matrices updated via the outer-product $v_t \otimes k_t$.

3.2. Alternative low-rank approximations

Kernel-based methods exploited to linearize Transformer self-attention are effective in reducing the computational cost with respect to the sequence length. However they result in a computational complexity quadratic to the model's feature dimension, $\mathcal{O}(L{d_{\text{k}}}^2)$, caused by their reliance on matrix dot products, which is inefficient forlarge model sizes (Zhai et al., 2021).

Attention-free Transformers A recent alternative emerged from a different low-rank factorization of the $\text{sim}(\cdot,\cdot)$ function in Eq. (7), leveraging intuitions which are related to Linear Attention of Eq. (9), even if based on element-wise multiplications that preserve the feature-dimension, i.e., $\text{sim}(q, k) = \sigma(q) \odot \psi(k)$. Attention Free Transformers (AFT) (Zhai et al., 2021) implement $\sigma(\cdot)$ as an elementwise nonlinear activation function (e.g., sigmoid) and $\psi(k) = e^k$, and perform the following operation,

$$y_t = \sigma(q_t) \odot \frac{\sum_{i=1}^{L} e^{k_i + w_{t,i}} \odot v_i}{\sum_{i=1}^{L} e^{k_i + w_{t,i}}}, \tag{16}$$

where the division is intended to operate element-wise, while $w_{t,i}$ denotes the (t, i)-th element of matrix $\mathcal{W} \in \mathbb{R}^{L \times L}$, which is a learnable matrix of pairwise position biases. In fact, for each input token at position t, AFT computes a weighted average of values, the result of which is combined with the query by an element-wise multiplication. This approach eliminates the need for dot product self-attention, while still preserving global interactions between query and values , and avoids the need to compute and store the attention matrix. As noted by the authors, AFT triggers an implicit attention mechanism with as many heads as the feature dimensions, wherein the attention matrices have a factorized structure. Such a procedure has a memory complexity that is linear

with respect to both the context size and the number of features, making it well-suited for both large inputs and large model sizes, and is generally referred to as *channel directed attention*. The special configuration in which no position biases are learned, referred to as AFT-simple, can be trivially formalized as

$$y_t = \sigma(q_t) \odot \frac{\sum_{i=1}^{L} e^{k_i} \odot v_i}{\sum_{i=1}^{L} e^{k_i}}. \tag{17}$$

Masked/causal formulation can be obtained by limiting the upper range of the summations to the current time index t (instead of L). It is easy to see that AFT-simple completely gets rid of dot products (the "softmax" is independent on the output index t), which results in a complexity of $\mathcal{O}(Ld_k)$ rather than $\mathcal{O}(Ld_k^2)$.

Reinventing RNNs for the Transformer Era Whilst both AFT and Linear Transformers largely reduce the computational requirements of classic softmax self-attention, they do not match its performances in real-world tasks, such as language modeling at scale. Inspired by AFT, the Receptance Weighted Key Value (RWKV) model (Peng et al., 2023) was proposed, driven by the intuition of giving less importance to "older" embeddings, thus replacing AFT pair-wise position embeddings $w_{t,i}$'s with exponential decays. In detail, in RWKV each $w_{t,i}$ is implemented with a decay factor, i.e., $w_{t,i} := -(t-i)w$ where $w \in \mathbb{R}^{d_k}_{\geq 0}$. Then, the RWKV architecture is the outcome of stacking pairs of blocks, where each pair consists of a of *time-mixing* and a *channel-mixing* residual layers, respectively.

The *time-mixing* layer is based on the computation of the so-called *receptance* r_t, of the key vector k_t and of value v_t. Then, the output of the block, y_t, depends on all such elements, together with the decayed position embeddings,

$$W_r(\mu_r \odot u_t + (1-\mu_r) \odot u_{t-1}), \tag{18}$$

$$W_k(\mu_k \odot u_t + (1-\mu_k) \odot u_{t-1}), \tag{19}$$

$$W_v(\mu_v \odot u_t + (1-\mu_v) \odot u_{t-1}), \tag{20}$$

$$W_o\sigma(r_t) \odot \frac{\sum_{i=1}^{t-1} e^{-(t-i-1)w+k_i} \odot v_i + e^{p+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-i-1)w+k_i} + e^{p+k_t}}, \tag{21}$$

where $\mu_\cdot$, $W_\cdot$, and p are additional trainable parameters. In Eq. (18)–(20), recurrence is implemented by means of a linear interpolation between input at t and $t-1$, referred to as *token-shift*. In Eq. (21), receptance r_t participates in the computation of a gate that acts as a forget gate, which eliminates unnecessary historical information. The recurrent nature of the model goes beyond Eq. (18)–(20), and it also affects Eq. (21), once we consider autoregressive decoding at inference. In fact Eq. (21) can be written in the following recurrent form,

$$\begin{aligned} s_t &= e^{-w} \odot s_{t-1} + e^{k_t} \odot v_t, \\ z_t &= e^{-w} \odot z_{t-1} + e^{k_t}, \\ y_t &= W_y\sigma(r_t) \odot \frac{s_{t-1} + e^{p+k_t} \odot v_t}{z_{t-1} + e^{p+k_t}}, \end{aligned} \tag{22}$$

where the *state* is $x_t := (s_t, z_t)$. The dataflow of the RNN-like time-mixing is shown in Fig. 7 (bottom). In AFT, $\mathcal{W}$ is a matrix of (pair-wise) position biases, while here it represents a channel-wise[8] vector multiplied by relative positions. Going beyond the just described *time-mixing* layer, RWKV, features a *channel-mixing* block, that computes its own receptance r'_t and keys k'_t following the same formulation of Eqs. (18) and (19), respectively (with separate learnable parameters). Then, the channel-mixing block computes its output y_t by means of $y_t = \sigma(r'_t) \odot (W'_v \cdot \mathrm{ReLU}^2(k'_t))$, leveraging the squared ReLU activation (So et al., 2021).[9] Noticeably, the token-shift operation allows the model to learn the amount of new information that should be stored into each channel of receptance, key, value and gate, resulting in the capability

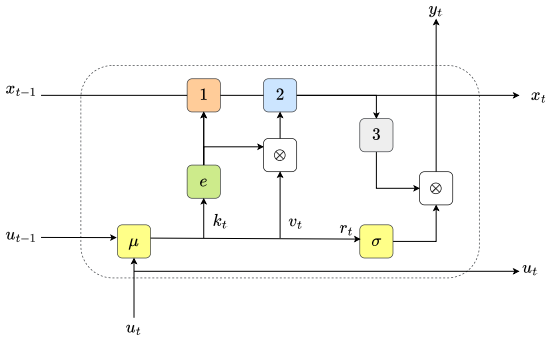

Fig. 7. RWKV time-mixing block (Peng et al., 2023). The yellow block (μ) implements the token shift; referring to Eq. (22), the orange block (1) denotes the computation of s_t and the blue block (2) denotes the computation of z_t (recall that x_t is the tuple (s_t, z_t)). The gray block (3) evaluates the fraction in the computation of y_t. See Peng et al. (2024a) for the recurrent form of later models (RWKV-5/6).

to compose induction heads within a single layer (Elhage et al., 2021). Indeed, though this formulation, a single head can directly accumulate both past and current token information into separate subspaces. The parallel (non-causal) form of RWKV has a time complexity of $\mathcal{O}(Ld_k^2)$. Updating attention scores in RWKV requires a serial scan (hence, the model cannot be parallelized along the sequence dimension) and has complexity of $\mathcal{O}(Ld_k)$. The simplification of dot-product attention to element-wise operations leads to significant efficiency gains, but at the same time it limits the model capability of capturing long-term dependencies.

Evolution of RWKV In a subsequent work (Peng et al., 2024a), the authors proposed two evolutions of the model, referred to as Eagle (or RWKV-5) and Finch (or RWKV-6). The former improves upon the previous architecture through the use of expressive multi-headed matrix-valued states (as opposed to vector-valued states), a reformulated receptance, and an additional gate g_t. For ease of notation, hereinafter the linear interpolation implementing the token-shift operation, Eqs. (18)–(20), will be more compactly defined by means of the lerp operator, $\mathrm{lerp}(a, b, \mu) := a + (b-a) \odot \mu$, where μ is a learnable vector. Moreover, we are in a multi-head setting, involving h heads, thus all the vectors belongs to $\mathbb{R}^{d_k/h}$. The channel-mixing block is exactly the same of the previous model, while the time-mixing block in RWKV-5 revises the one in Eqs. (18)–(21) as follows,

$$\begin{aligned} r_t &= W_r\mathrm{lerp}(x_t, x_{t-1}, \mu_r), \quad k_t = W_k\mathrm{lerp}(x_t, x_{t-1}, \mu_k), \\ v_t &= W_v\mathrm{lerp}(x_t, x_{t-1}, \mu_v), \quad g_t = W_g\mathrm{lerp}(x_t, x_{t-1}, \mu_g), \\ w &= \exp\big(-\exp(\omega)\big), \\ \mathrm{WKV}_t &= \mathrm{diag}(u)(v_t \otimes k_t) + \sum_{i=1}^{t-1} \mathrm{diag}(w)^{t-1-i}(v_i \otimes k_i), \\ y_t &= W_o\mathrm{concat}\big(\mathrm{SiLU}(g_t) \odot \mathrm{LayerNorm}(\mathrm{WKV}_t r_t)\big), \end{aligned} \tag{23}$$

where $\mathrm{WKV}_t \in \mathbb{R}^{(d_k/h)\times(d_k/h)}$ is the attention score matrix, the $\mathrm{concat}(\cdot)$ operation concatenates the contributions from different heads, $\mathrm{diag}(\cdot)$ yields a square matrix whose diagonal is the fuction argument, while LayerNorm (layer normalization) operates on each of h heads separately. Please notice that w is obtained from a double exponentiation where $\omega \in \mathbb{R}^{d_k/h}$ are headwise trainable parameters, thus ensuring that $w \in (0,1)$, and guaranteeing that $\mathrm{diag}(w)$ is a contraction matrix. It turns out that the attention score WKV_t in the Eagle model can be expressed in the recurrent form,

$$\begin{aligned} S_t &= \mathrm{diag}(w)S_{t-1} + v_t \otimes k_t, \\ \mathrm{WKV}_t &= S_t + \mathrm{diag}(u)(v_t \otimes k_t), \end{aligned} \tag{24}$$

confirming the recurrent nature of this model as well. From such a recurrent form, it is easy to notice that the state S_t is a sum over outer products where each channel is individually decayed by the corresponding

[8] The term channel refers to the feature dimension.

[9] Implemented as $\mathrm{ReLU}^2(x) := \big(max(0,x)\big)^2$

channel of w, at each time step. The attention score of Eq. (24) (bottom) is computed by applying a per-channel boost u, multiplied with the current token's outer product $v_t \otimes k_t$, giving the current token a special treatment relative to the sum of past tokens contained within the decaying state history. The Finch/RWKV-6 model further improves the architecture expressivity by injecting data-dependence for both the time-mixing and token-shift modules. Additionally, it proposes to leverage Low Rank Adaptation (LoRa) (Hu et al., 2021) to effectively exploit the learned data decay vectors in a context-specific manner. Indeed, the token shift in Finch leverages an augmented data-dependent linear interpolation ddlerp() implemented as follows,

$$\text{LoRa}(x, A, B, \lambda) = \lambda + B\tanh(Ax),$$

$$\text{ddlerp}(a, b, A, B, \lambda) = a + (b - a)\odot \text{LoRa}(a + (b - a) \odot \mu_x, A, B, \lambda),$$

where $B \in \mathbb{R}^{d_{\text{in}} \times 32}$, $A \in \mathbb{R}^{32 \times d_{\text{in}}}$, and $\mu_x, \lambda \in \mathbb{R}^{d_{\text{in}}}$ are trainable parameters. In this novel form of data-dependent token-shift the amount of new and old data allocated per channel now depends on the input at both current and prior time steps. The time-mixing block extends Eq. (23) as follows (replacing the equation of w and of WKV_t),

$$d_t = \text{LoRa}(\text{ddlerp}(x_t, x_{t-1}, A, B, \lambda), A, B, \lambda),$$

$$w_t = \exp\big(-\exp(d_t)\big),$$

$$\text{WKV}_t = \text{diag}(u)(v_t \otimes k_t) + \sum_{i=1}^{t-1} \text{diag}(\odot_{j=1}^{i-1} w_j)(v_i \otimes k_i).$$

Differently from Eagle/RWKV-5 where w is a fixed vector, in Finch/RWKV-6 each channel of w_t varies over time in a data-dependent manner. The LoRa operators allows to inexpensively augment learned vectors with additional offsets determined by the incoming input.

3.3. Modeling recurrence

Amidst the multiple advantages brought by the vanilla self-attention mechanism, early attempts to tackle language modeling with Transformers were hampered by the inability (i) to model intra-sequential relations due to the static order dependencies available in standard positional encodings (i.e., the absence of temporal information), and (ii) to propagate inter-sequence information among processed contexts. Several attempts to tackle these two issues are presented in the following. Additionally, such approaches inspired recent sub-quadratic methods that split the overall computation into sequence-chunks that are processed in parallel.

Intra-sequence Recurrence Modeling An interesting line of work leverages recurrent mechanisms to increase the representational power of features extracted by Transformers, e.g., by injecting locality biases or temporal ordering in the obtained outputs. Chen et al. (2018a) showed that representations learned by RNN-based encoders can be augmented by those learned with a self-attention encoder, resulting in an improvement in performance for RNN-based neural machine translation (NMT) tasks (Stahlberg, 2020). Inspired by this work, Hao et al. (2019) propose to directly model recurrence in an encoder-decoder Transformer architecture for NMT. This is done by leveraging an additional recurrence encoder, $E_{\text{rec}}(\cdot)$, that operates in parallel with respect to the standard Transformer encoder, hereinafter referred to as $E(\cdot)$. The authors propose to implement $E_{\text{rec}}(\cdot)$ as (i) a bidirectional RNN or as (ii) an Attentive Recurrent Network (ARN), a model that performs recurrent steps on the features vectors extracted with an attention model from the input representations U. For a generic layer l, and its input U^l, ARN computes:

$$x_t^l = F(x_{t-1}^l, c_t^l),$$

$$c_t^l = \text{ATT}(x_{t-1}^l, U^{l-1}),$$

where $F(x_{t-1}^l, c_t^l)$ denotes an RNN-like transition function (e.g., such as Eq. (2) or the one used in LSTM) with state x_t, which processes an external input value c_t. The operation $\text{ATT}(x_{t-1}^l, U^{l-1})$ is an attention procedure over the layer-input representations U^{l-1}, exploiting the previous state x_{t-1}^l as query. Outputs from $E(\cdot)$ and $E_{\text{rec}}(\cdot)$ are fused either by a gating mechanism or by sequentially attending them in the decoder. Injecting this kind of recurrent mechanisms into purely attention-based architectures has been proven to boost their performance. Chen et al. (2019) attribute these results on the fact that positional embeddings exploited by standard Transformer architectures are based solely on discrete numerical indices (e.g., using trigonometric functions (Vaswani et al., 2017)) and are independent of word content. Hence, they are not prone to capture semantic dependencies between words in a sentence. To overcome this issue, the authors in Chen et al. (2019) split the embeddings of each input word/token u_t into two parts, resulting in two input sequences $U^p = (u_t^p)_{t=1}^L$ and $U^r = (u_t^r)_{t=1}^L$. Then, they explicitly learn recurrent positional embeddings (RPEs) on U_r with a non-linear RNN, i.e., RPEs are the elements of the temporal sequence of states computed by such RNN. At the same time, the other sub-sequence U^p is leveraged to compute positional word embeddings following the standard Transformer pipeline. The concatenation of such embeddings and RPE is then given as input to the Transformer encoder (or decoder), equipped with ad-hoc heads to process the recurrent positional information. Leveraging such recurrent embeddings allows the model to capture order-related dependencies.

Huang et al. (2022) proposed a block-diagonalization of a linear RNN layer that allows to rewrite it into a lightweight relative positional encoding matrix for a multi-head self-attention, named Recurrent Encoding Matrix (REM). The overall intuition is to encapsulate the recurrent dynamics of the RNN layer into the positional encodings of a multi-head attention layer, leading towards a self-attention with recurrence (RSA). In particular, by considering a *linear* RNN (obtained from Eq. (2) when σ is the identity function), it is possible to write it in the following compact form,

$$x_t = Ax_{t-1} + Bu_t = \sum_{j=0}^{t-1} A^j B u_{t-j}. \tag{25}$$

Thus, the authors propose to block-diagonalize the A matrix such that the RNN can be broken down into a sequence of simpler RNNs. Under mild assumptions, A has r real non-zero eigenvalues $(\lambda_1, \dots, \lambda_r)$ and s pairs of complex nonzero distinct eigenvalues (the pair $(\gamma_k e^{i\theta_k}, \gamma_k e^{-i\theta_k})$ with $1 \le k \le s$ where i is the imaginary unit). The corresponding real Jordan form is $A = G\Lambda G^{-1}$, where $G \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ is invertible and $\Lambda \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ is a block diagonal matrix. The exponentiation of this matrix is easy to compute, i.e., $A^j = G\Lambda^j G^{-1}$, and it is possible to break down the recurrence induced by A into that of $p \times p$ block matrices in Λ, with $p \in (1, 2)$. As a result, the linear RNN layer can be decomposed into three different contributions, namely $(x_t^R, x_t^{C1}, x_t^{C2})$, the first one corresponding to real eigenvalues (the 1×1 blocks in Λ) and the last two to the complex ones (the 2×2 blocks in Λ). This allows to rewrite Eq. (25) as $x_t = \sum_{k=1}^r x_t^R(\lambda_k) + \sum_{k=1}^s x_t^{C1}(\gamma_k, \theta_k) + \sum_{k=1}^s x_t^{C2}(\gamma_k, \theta_k) + Bu_t$. The first term is defined as follows $x_t^R(\lambda_k) := \sum_{j=1}^{t-1} \lambda^j B^R u_{t-j}$. See the referenced paper for the complete forms of x_t^{C1} and x_t^{C2}. When considering the input matrix U, it is interesting to see that the aforementioned decomposition allows to write the RNN as a multihead self-attention with $r + 2s + 1$ heads, with null query and values and where the positional encodings matrices encapsulate the recurrent dynamics. In details,

$$(x_1^*, \dots, x_T^*) = \big(\text{softmax}(QK') + P^*(\lambda)\big)V,$$

that is differently instantiated for $x_t^* \in (x_t^R, x_t^{C1}, x_t^{C2})$. The same holds for $P^*(\lambda_k)$, which is a relative positional encoding (lower triangular in the causal masked case) matrix, referred to as Recurrence Encoding Matrix (REM). For instance, when considering x_t^R, $V = UB^R$ and $P^R(\lambda_k)$ has a specific form (see the referenced paper for the case of P^{C1} and P^{C2}). These three REMs, P^R, P^{C1}, P^{C2} summarize different temporal decays patterns: the regular REM, P^R, provides the regular exponential decay induced by the real eigenvalues. Cyclical REMs, (P^{C1}, P^{C2}), provide the cyclical damped cosine or sine decay induced by the pair of complex eigenvalues. REM can be injected into any existing self-attention based

Transformer architecture, leading to the Self-Attention with Recurrence (RSA) module,

$$RSA(U) = \Big(\big(1 - \sigma(\mu)\big)\text{softmax}(QK') + \sigma(\mu)P^* \Big)V,$$

where $\sigma(\mu) \in [0, 1]$ is a gate parametrized by μ, σ the sigmoid function and P^* is a regular or cyclical REM. RSA models the token-level recurrence, which is at the most finegrained scale. Subsequently, it can be easily incorporated into the coarser-grained designs that will be the subject of the next paragraph, and may potentially bring benefits to their performance. Token Turing machines (Ryoo et al., 2023) takes the alternative route of exploiting memory-based mechanisms, inspired by Neural Turing Machines (Graves et al., 2014). An external memory, populated with token summarizations, is read/written using a Transformer as the processing unit/controller at each step. Hence, constant compute is achieved, regardless of the length of the history, hence resembling the computational mechanism of recurrent models where the memory is the neural state.

Segment-level Recurrences Standard Transformer training is performed on separate fixed-length segments of text, derived from the context window span, without any information flow across such segments, resulting in the inability to capture any longer-term dependency beyond the predefined context. Hence, the ability to temporally connect different contexts becomes extremely important. The basic feature that can overcome such limitations consists of maintaining a cache memory, to be read as an additional input, in order to store the state of previously processed information. This can be easily implemented by exploiting a segment-level recurrence to aggregate the temporal information at a coarser scale, while the token-by-token dependencies are learned by self-attention as usual (Lin et al., 2022). A seminal work in this direction is Transformer-XL (Dai et al., 2019), which sequentially processes consecutive segments having size T, exploiting a segment-level recurrence. Layer-wise representations from previous segments are cached and exploited as an extended context for the current segment. Considering the l-th layer of a Transformer architecture and the s-th segment, we denote with U_s^l the l-th layer input representation of the s-th input segment (i.e., it corresponds to U_s when $l = 0$ and to the output of the previous Transformer layer for the segment s, i.e., Y_s^{l-1}, when $l > 0$). In Transformer-XL, U_s^l is concatenated with the representations from the previous segments, Y_{s-1}^{l-1}, to compose a state that is exploited to produce keys and values as follows,

$$\begin{aligned}
&X_s^l = [\text{sg}(Y_{s-1}^{l-1}) \mid U_s^l], \\
&Q_s^l = X_s^l W_q, \quad K_s^l = X_s^l W_k, \quad V_s^l = X_s^l W_v, \\
&Y_s^l = U_s^{l+1} = \text{tr}^l(Q_s, K_s, V_s),
\end{aligned}$$

where $\text{sg}(\cdot)$ denotes an operation (i.e., stop-gradient) which avoids the gradient flow during Backpropagation, $[\cdot|\cdot]$ denotes concatenation and $\text{tr}^l(\cdot)$ the l-th Transformer layer. Notice that the recurrent dependency shifts one-layer downward per segment, as depicted in Fig. 8, which differs from the same-layer recurrence in conventional RNNs. Thus, the largest possible dependency length grows linearly w.r.t. the number of layers as well as the segment length. Additionally, it is possible to not limit the state-cache solely to the previous segment, by storing the last m states and using them as the extra context when processing the current segment. Thus, a Transformer-XL with N layers and with a memory of size m, has a maximum temporal range of $N \times m$ with a memory complexity in the attention operation of $\mathcal{O}(T^2 + Tm)$ when processing a segment with length T.

Transformer-XL inspired a plethora of model variants (Ding et al., 2021; Lei et al., 2020; Rae et al., 2019; Wu et al., 2022a). Compressive Transformer (Rae et al., 2019) extends the cache with two levels of memory, exploited to store compressed older activations. Memformer (Wu et al., 2022a) extends the recurrence mechanism from decoder-only architecture to an encoder-decoder architecture. R-Transformer (Wang et al., 2019) inputs are first fed to a local RNN, that captures local dependencies, and then to multi-head self-attention module. Please refer to the survey (Lin et al., 2022) (mostly Section 6.4.1) for an overview of these methods. Recurrent Memory Transformer (RMT) (Bulatov et al., 2022) propose an alternative recurrent approach, where the input segment is augmented with m real-valued *memory* tokens, M_s, added both at the beginning and at the end of the input segment U_s, as follows,

$$\begin{aligned}
&\hat{U}_s = [M_s|U_s|M_s], \\
&\hat{Y}_s = \text{tr}(\hat{U}_s),
\end{aligned}$$

where the positions in the Transformer output sequence corresponding to the memory M_s are interpreted as a read/write memory, i.e., when considering an N-layered architecture, the output of the multilayer transformer can be interpreted as partitioned as follows, $\hat{Y}_s := [M_s^{\text{read}}|Y_s^N|M_s^{\text{write}}]$ (see also Fig. 8). The *memory* tokens play a two-fold role: the ones placed at the sequence start allow the standard sequence tokens to attend to memory states produced at the previous segment. The ending group acts as a write memory, that updates its representation based on all the current segment tokens. As a result, M_s^{write} contains updated memory tokens for the s-th segment. Recurrent connection between segments is achieved by passing the memory tokens from the current segment as the input to the next segment,

$$M_{s+1} := M_s^{\text{write}}, \quad \hat{U}_{s+1} = [M_{s+1}|U_{s+1}|M_{s+1}].$$

In RMT the Effective context length for recurrence with memory is not limited by the depth of the network, which is the case for the cache of Transformer-XL. Moreover, while Transformer-XL stores $m \times T$ vectors per segment, RMT stores only m memory vectors per segment. RMT is trained with BPTT, and the number of segments to backpropagate is a hyperparameter of the training procedure (the authors tested from 0 to 4 segments). Differently from Transformer-XL, during the backward pass memory-related gradients are not stopped between segments. Fig. 8 de-

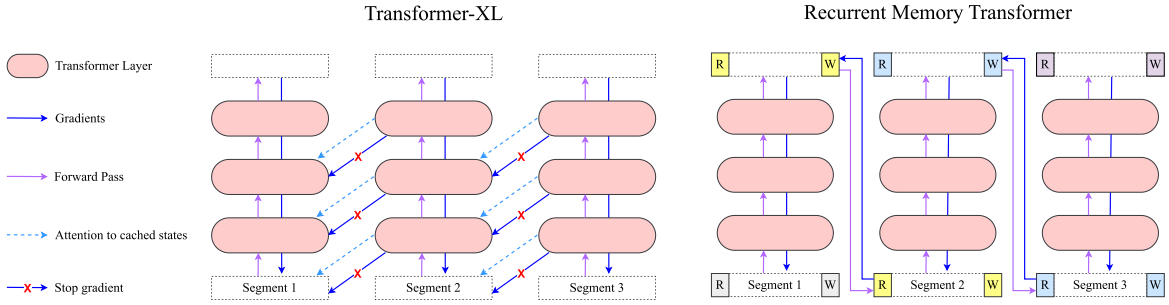

Fig. 8. Comparison of Transformer-XL and Recurrent Memory Transformer (RMT) architectures. Recurrent Memory Transformer augments Transformer with global memory tokens, allowing a segment-level recurrence. Special read/write memory tokens (denoted in the figure with R and W for ease of notation – in the main text they are M^{read} and M^{write}, respectively) are added to the input sequence. Multiple memory tokens can be used in each read/write block. Updated representations of write memory are passed to the next segment. During training, RMT uses BPTT to propagate gradient to previous segments through memory tokens representation. Effective context length for recurrence with memory is not limited by the depth of a network which is the case for the cache of Transformer-XL..

picts the architectural differences between Transformer-XL and RMT. In a recent technical report (Bulatov et al., 2023), the authors leverage the RMT architecture to extend the context length of a BERT model (Kenton & Toutanova, 2019) up to two million tokens, while maintaining high memory retrieval accuracy. Another approach that still falls in the category of models described so far is Infini-Transformer (Munkhdalai et al., 2024), which incorporates a compressive memory into a vanilla attention mechanism. It also builds in both causal local attention and long-term linear attention mechanisms into a single Transformer block. Inspired by Dai et al. (2019), the authors remark that performing attention solely on the current segment (that can be seen as a form of *local attention*) discards the attention states of the previous one. To counteract this issue, they propose to maintain the entire context history in the form of a compressive memory. An Infini-Transformer layer contains both global compressive and local fine-grained states. In fact, it maintains as many parallel compressive memories as the number of attention heads, in addition to the dot-product attention. Each segment is processed via a classic softmax self-attention, producing an output Y_{loc}. Then, a linear attention (see Eq. (9)) is exploited to retrieve content from the previous segment memory/state, M_{s-1}, as follows,

$$Y_{\text{mem}} = \frac{\phi(Q)M_{s-1}}{\phi(Q)z_{s-1}},$$

where $\phi(x) = \text{elu}(x) + 1$, s is an index over the segments and $z_{\cdot}$ is the normalization factor of linear attention (see Eq. (9), here we used the parallel form, in matrix notation). State/memory update is implemented by a delta-rule-like mechanism (Schlag et al., 2021) (see Section 3.1), first retrieving existing entries (values) and subtracting them from the new values, before applying the associative bindings, as follows,

$$M_s = M_{s-1} + \phi(K)'\left(V - \frac{\phi(K)M_{s-1}}{\phi(K)z_{s-1}}\right),$$

$$z_s = z_{s-1} + \sum_{t=1}^{T}\phi(K_t),$$

where T denotes the segment length. The new memory states M_s and z_s are then passed to the next segment $s+1$, building in a recurrence in each attention layer. Finally, local attention Y_{loc} and memory retrieved content Y_{mem} are aggregated via a learned gating mechanism, weighted by a learnable scalar β, i.e., $Y = \sigma(\beta)Y_{\text{mem}} + (1-\sigma(\beta))Y_{\text{loc}}$.

Chunk-level Recurrences The recently introduced segment-based approach allows networks to process very long texts sequentially, keeping a recurrent memory/context. Another category of emerging approaches involving recurrent models that handle portions of text consists of processing sub-portions of the input sequence, referred to as *chunks*, by dividing the sequence into non-overlapping chunks and performing (serial) inter-chunk recurrent computations followed by (parallel) intra-chunk computations. Such a *chunk-wise parallel form* yields sub-quadratic complexity. Temporal Latent Bottleneck (TLB) (Didolkar et al., 2022) divides the input sequence $U = (u_1, \dots, u_L)$ into chunks of fixed size C, resulting in $\lfloor L/C \rfloor$ chunks that are sequentially processed one after the other. We denote the i-th chunk of the input sequence as $U_{[i]} := U_{iC:(i+1)C} \in \mathbb{R}^{C \times d_{\text{in}}}$. Each chunk is processed by a fast-stream (also referred to as *perceptual* module), implemented by a Transformer $\text{tr}(\cdot)$. The fast-stream computation is conditioned via cross-attention on information coming from a slow-stream, referred to as Temporal Latent Bottleneck $\mathcal{G}$, that aggregates cues across chunks and is updated once per chunk. Such a slow-stream is computed recurrently, and it manages a state X composed of a set of N d_s-dimensional vectors. The state update in $\mathcal{G}(\cdot)$ is performed via a cross-attention operation in which the queries are obtained by projecting X, while the keys and values come from the output of the perceptual module. Overall, the model performs the following operation,

$$\hat{Y}_{[i]} = \text{tr}(U_{[i]}, X_{[i]}),$$
$$X_{[i+1]} = \mathcal{G}(X_{[i]}, \hat{Y}_{[i]}).$$

The recurrent update of the TLB state X is performed at lower rates with respect to the computations happening in the perceptual module, fostering the distillation of more stable, condensed and slowly changing features, while the perceptual module captures faster changing local cues. From the computational point of view, leveraging a chunk-based approach allows to achieve a complexity of $\mathcal{O}\left(\frac{L}{C}(C^2 d + CN)\right)$, where N is the number of temporal latent bottleneck state vectors. Hence, it has a much lower computational complexity compared to a vanila Transformer applied on the overall sequence. A concurrent work, Block-Recurrent Transformer (Brect) (Hutchins et al., 2022), handles the input sequence in a similar manner to Transformer-XL: a document is split into multiple segments of size T, processed one after the other. Each segment is processed in chunks (or blocks) using a sliding window attention with size C, with a causal mask that forces each token to attend solely to the previous ones. Brect is composed of a recurrent cell that receives as inputs C token embeddings, where C is the block/window size, and a set of N *state* vectors. Similarly to TLB, a two-way processing is performed in the proposed recurrent cell, consisting of the so-called vertical and horizontal "directions": the *vertical* direction (i.e., corresponding to the fast-stream in TLB) is implemented by a Transformer layer that performs self-attention over the input tokens of the current block and cross-attends to the recurrent states, producing output token embeddings for the current block. The *horizontal* direction (i.e., the slow-stream in TLB) performs self-attention over the current state vectors, and cross-attends to the block input tokens, producing the next state vectors. Unlike TLB, cross-attention and self-attention are computed in parallel. Recurrence is integrated with the sliding window attention mechanism, since keys and values from the previous window are stored in a differentiable cache and concatenated to the ones of the current block. Residual connections are replaced by gating mechanisms (implemented either as fixed convex combinations or trainable LSTM-like gates) that help the model forget redundant information. For every layer of the architecture, the last recurrent states of a segment are stored in a non-differentiable cache and fed to the following segment in the document as a warm-start. This mechanism extends the sliding window to cover the entire sequence. The cache implements a form of truncated BPTT over long documents. Block-State Transformer (Pilault et al., 2023) replaces the recurrent cell in Brect with a linear state-space model (see Section 4), which processes the sequence and preserves long-range dependencies, while allowing also for parallel computations.

Recurrent Attention Networks (RAN)(Li et al., 2023b) iteratively process the input sequence by means of non-overlapping windows, each of them processed via multi-head self-attention. Intra-window information is propagated by a global perception cell (GPC) vector, extracted from the self-attention representation of the current window. The GPC vector is concatenated with the input tokens of the next window. A memory review mechanism cross-attends the concatenation of the GPC vectors of all the windows to build the final output, encoding both contextual and global information. We report in Fig. 9 the main differences in the processing schemes of segment level and chunk-level approaches, reporting examples of some of the described models.

3.4. Decaying linear attention by gating functions

Gating mechanisms were introduced since the dawn of RNNs, being a key feature of the popular LSTM model (Hochreiter & Schmidhuber, 1997; Qin et al., 2023c). In previous subsections and in the context of linear approximations to attention mechanisms, we described several works connecting Linear Transformers to Fast weight Programmers (FWP), which introduced decay/gating mechanisms inherited from FWP (Irie et al., 2021, 2022b; Mao, 2022; Schlag et al., 2021). The original gating functions in LSTMs consisted of units with sigmoidal activations, responsible for gating specific signals (input, output, state) via multiplicative operations. Lately, the concept of gating has been relaxed to consider any multiplicative interaction, possibly followed by an acti-

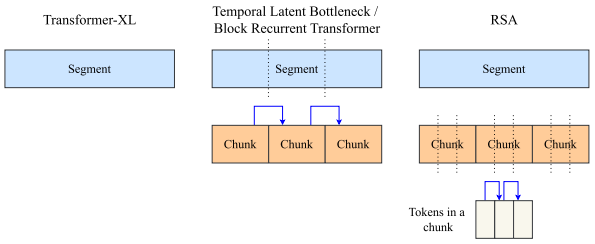

Fig. 9. From segment-level to chuck-level recurrence. From left to right: Transformer-XL implements the most coarse-grained segment-level recurrence. Differently, Temporal Latent Bottleneck and Block-Recurrent Transformers include finer-grained chunk-level (or block-level) recurrence. RSA implements the most fined-grained token-level recurrence.

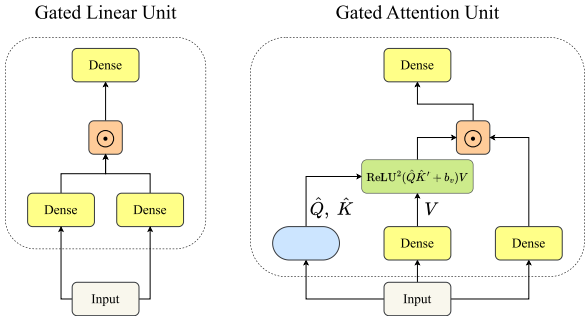

Fig. 10. Gated Linear Unit (left) and Gated Attention Unit (right).

vation function (e.g., element-wise multiplicative components that do not interact along the sequence length are referred to as gating mechanisms (Hua et al., 2022; Mehta et al., 2022)). Nevertheless, the role of gating has become increasingly popular and now pivotal in many works.

Multiplicative Interactions Hua et al. (2022) remarked that despite the linear theoretical complexity claimed by linear attention methods (Katharopoulos et al., 2020), they have not been able to supersede vanilla Transformers as the dominant choice in state-of-the-art systems. They attributed this to (*i*) approximations needed to achieve efficiency; (*ii*) non-trivial gap between theoretical complexity and empirical speed on accelerators, due to memory re-formatting and operations (see also Section 8.2); (*iii*) slow training on causal autoregressive tasks, due to sequential processing of the recurrent forms. To counteract these issues, the authors proposed FLASH, which is based on a novel layer, dubbed Gated Attention Unit (GAU), to substitute softmax multi-head self-attention. The key idea is to combine a Gated Linear Unit (GLU)[10] (Dauphin et al., 2017) and attention in a unified layer, as depicted in Fig. 10.

The parallel form of GAU generalizes a GLU as follows,

$$\begin{aligned} &R = \sigma(UW_r), \quad V = \sigma(UW_v), \quad Z = \sigma(UW_z), \\ &\hat{Q} = \mathcal{Q}(Z), \quad \hat{K} = \mathcal{K}(Z), \quad \hat{V} = \mathrm{ReLU}^2(\hat{Q}\hat{K}' + b_v)V, \\ &Y = (R \odot \hat{V}W_y), \end{aligned} \tag{26}$$

where $W_r, W_v \in \mathbb{R}^{d_{\text{in}} \times d_e}$, $W_z \in \mathbb{R}^{d_{\text{in}} \times d_z}$ with $d_z \ll d_{\text{k}}$, $\mathcal{Q}, \mathcal{K}$ are learned transformations that apply per-dim scalars and offsets to Z, $W_y \in \mathbb{R}^{d_e \times d_y}$ denotes an output learnable matrix, and $b_v \in \mathbb{R}^{d_z}$ are relative position bias. Notice that this formulation substitutes the softmax with the squared ReLU (So et al., 2021) (denoted with ReLU^2). This single-headed softmax-free formulation is capable to achieve multi-head Transformers performances without quality loss, and is cheaper in terms of computational requirements, by adding solely the learnable matrix W_z with $d_{\text{in}} \times z$ parameters on top of the GLU. Additionally, the authors analyze the causes underlying the aforementioned memory and speed issues (issues *ii* and *iii*). In fact, despite the huge advantages brought by the constant-inference computation and memory cost in autoregressive decoding (i.e., thanks to the stateful representation S_t) the rearrangement of computations in linear attention lead to an inefficiency in the case of autoregressive training.[11] To counteract this, FLASH provides a mixed-chunk attention formulation, where the input sequence is chunked into $\lfloor L/C \rfloor$ chunks with fixed size C. From each chunk c, representations $R_{[c]}, V_{[c]} \in \mathbb{R}^{C \times d_e}$ and $Z_{[c]} \in \mathbb{R}^{C \times d_z}$ are obtained following Eq. (26). Then, an attention mechanism composed by a local (quadratic) and a global (linear) component is formulated. Local attention follows the same procedure of Eq. (26) to compute a local chunk-based version of $\hat{V}$, denoted with $\hat{V}^{\text{loc}}_{[c]}$, given by $\hat{V}^{\text{loc}}_{[c]} = \mathrm{ReLU}^2(\hat{Q}^{\text{loc}}_{[c]}\hat{K}^{\text{loc}\prime}_{[c]})V_{[c]}$. Here $\hat{Q}^{\text{loc}}_{[c]}$ and $\hat{K}^{\text{loc}}_{[c]}$ are the outcome of two ad-hoc per-dim scaling/offsets transformations of $Z_{[c]}$. The complexity of these operations is linear in the sequence length, i.e., $\mathcal{O}(LCd_{\text{in}})$. Differently, the global linear attention captures long-interactions across chunks, exploiting other ad-hoc per-dim scaling and offsets transformations of $Z_{[c]}$, $\hat{Q}^{\text{glob}}_{[c]}$, and $\hat{K}^{\text{glob}}_{[c]}$. The causal formulation is defined as follows,

$$\hat{V}^{\text{glob}}_{[c]} = Q^{\text{glob}}_{[c]} \left(\sum_{h=1}^{[c]-1} \hat{K}^{\text{glob}\prime}_{[c]} V_h \right). \tag{27}$$

Summation is performed at the chunk level, reducing the number of elements in the cumulative sum by a factor of C. The local and global contributions, $\hat{V}^{\text{loc}}_{[c]}, \hat{V}^{\text{glob}}_{[c]}$, are added together to yield the final output, $Y_{[c]} = R_{[c]} \odot (\hat{V}^{\text{glob}}_{[c]} + \hat{V}^{\text{loc}}_{[c]})]W_y$. For autoregressive training, thanks to chunking, the sequential dependency in the auto-regressive case reduces from L steps in the standard linear attention to L/C steps in the chunked version in Eq. (27). Another work focussing on gating functions with multiplicative interactions is the so-called Moving-average Equipped Gated Attention mechanism (MEGA) (Ma et al., 2023), which injects a temporal locality inductive bias into the attention mechanism by leveraging a multidimensional exponential moving average (EMA) (Hunter, 1986). The EMA captures local dependencies that exponentially decay over time, and is then integrated with a variant of the single-head GAU. The multidimensional damped EMA firstly expands each dimension of the input sequence U into d_{s} dimensions via an expansion matrix $\beta \in \mathbb{R}^{d_{\text{in}} \times d_{\text{s}}}$, to increase the expressiveness of the model, i.e. $\hat{U} = U\beta$. Then, the EMA update with state $x_t \in \mathbb{R}^{d_{\text{s}}}$ is carried on as follows,

$$\begin{aligned} x_t &= \alpha \odot \hat{u}_t + (1 - \alpha \odot \delta) \odot x_{t-1}, \\ y_t &= \eta' x_t, \end{aligned} \tag{28}$$

where $\delta \in (0,1)^{d_{\text{in}}}$ is a damping factor, $\alpha, \beta, \nu \in \mathbb{R}^{d_{\text{s}}}$, while $x_t \in \mathbb{R}^{d_{\text{s}}}$ is the EMA state at timestep t; $\hat{u}_t \in \mathbb{R}^{d_{\text{s}}}$ is the expanded input vector at time t (i.e., a column vector that is extracted from a row of $\hat{U}$), and η is a projection vector to map the d_{s}-dimensional hidden state back to 1-dimensional output. Despite the recurrent formulation in Eq. (28), computation of EMA can be represented as t individual convolutions, which can be computed efficiently using fast Fourier Transforms (FFTs) (see Section 4). As we will describe in the next Section, the multi-dimensional damped EMA can be seen as a simplified variant of a state-space model, and MEGA is closely related to S4 (Gu et al., 2021a), a state-space model with structured state matrices. The EMA sub-layer in MEGA applies diagonalization on the state matrix and restricts the diagonal elements

[10] A GLU (Dauphin et al., 2017) is an MLP which output is modulated via a gating (i.e., a learned multiplicative interaction). In fact, the layer input U is projected by two learnable matrices $W_p \in \mathbb{R}^{d_{\text{in}} \times d_e}$ and $W_t \in \mathbb{R}^{d_{\text{in}} \times d_e}$ into two representations $P, T \in \mathbb{R}^{L \times d_e}$, that interact in an element-wise multiplicative manner to produce the layer output, as follows:

$$R = \sigma(UW_p) \quad V = \sigma(UW_t) \quad Y = (R \odot V)W_y.$$

[11] Due to the causal constraint for auto-regressive training, the query vector corresponds to a different cache value at each time step. This requires the model to compute and cache L different values of the incremental state and requires L memory accesses in the sequential loop.

in the range of $(0, 1)$. Thus, the convolution kernel would be a Vandermonde product, which can be computed in an efficient and numerically stable way. The output from Eq. (28) is collected into a matrix $\hat{Y}$, which is propagated into a mixed GAU-GRU architecture. The former follows Eq. (26) to transform $\hat{Y}$ (the authors leverage a SiLU activation function (Ramachandran et al., 2017) instead of the ReLU^2). Then, the multiplicative interaction in the output inherited from GAU is combined with reset and update gates from GRUs. The authors additionally propose MEGA-chunk, a model variant with linear complexity due to its chunked-form, where the EMA component takes care of extending the effective context being exploited by chunk-wise attention.

Decaying Interactions Sun et al. (2023a) propose a *retention* mechanism in place of attention, based on an explicit decay matrix, that controls the ability of each token to pool information from its surrounding tokens based on distance priors. The proposed RetNet encodes the sequence autoregressively. Retention implements a sequence modeling problem in a recurrent fashion, by leveraging a linear recurrence with state $s_t \in \mathbb{R}^{d_s}$ and a scalar projection of the input, $v_t \in \mathbb{R}$, obtained via $v_t = u'_v u_t$, as follows,

$$\begin{aligned} s_t &= A s_{t-1} + k_t \cdot v_t, \\ y_t &= q'_t s_t = \textstyle\sum_{m=1}^{t} q'_t A^{t-m} k_m v_m, \end{aligned} \tag{29}$$

where $q_\cdot, k_\cdot$ are the usual queries and keys computed following Eq. (7) with projection matrices W_q, W_k. Eq. (29) (top) maps v_t onto the state vector s_t, and then implements a linear transformation to encode the sequence information recurrently (bottom). Matrix A is diagonalized into $A = \Lambda(\gamma e^{i\theta})\Lambda^{-1}$, with $\gamma, \theta \in \mathbb{R}^{d_k}$. Similarly to RSA (Huang et al., 2022), the exponentiation yields $A^{t-m} = \Lambda(\gamma e^{i\theta})^{t-m}\Lambda^{-1}$. By simplifying γ as a scalar and absorbing Λ into the projection matrices W_q, W_k, it is possible to simplify Eq. (29) as,

$$y_t = \sum_{m=1}^{t} \gamma^{t-m} (q'_t e^{it\theta})(k_m e^{im\theta})^{\dagger} v_m,$$

where $\dagger$ denotes the conjugate transpose. Notice the resemblance between the multiplying factors and the xPOS (Sun et al., 2023b) positional encodings.

Starting from this formulation, it is easy to obtain the RetNet *parallel form* (Fig. 11, left), which is defined as follows when considering a vector mapping instead of the scalar one of Eq. (29),

$$\begin{aligned} &Q = (UW_q) \odot \Theta, \quad K = (UW_k) \odot \overline{\Theta}, \quad V = UW_v, \\ &\Theta_t = e^{it\theta}, \qquad D_{tm} = \begin{cases} \gamma^{t-m}, & t \geq m \\ 0, & t < m \end{cases}, \\ &\text{Retention}(U) = (QK' \odot D)V, \end{aligned}$$

where $\overline{\Theta}$ is the complex conjugate of Θ, and $D \in \mathbb{R}^{L \times L}$ contains both causal masking and exponential decay, encoding the prior temporal knowledge as a relative distance in the one-dimensional sequence. This form is particularly advantageous for parallel training. The retentive mechanism can be directly written into a recurrent form by means of 2D-states, S_t, (Fig. 11, right),

$$\begin{aligned} S_t &= \gamma S_{t-1} + k_t \otimes v_t, \\ y_t &= S_t q_t. \end{aligned} \tag{30}$$

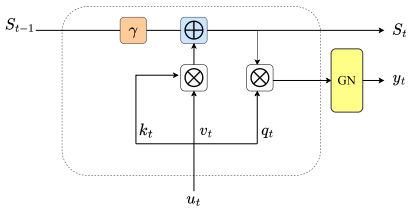

Fig. 11. RetNet, recurrent form (GN is short for GroupNorm).

It is easy to discern that Eq. (30) corresponds to Eq. (15) with the addition of the fixed decay factor γ, which is usually selected as $\gamma = 1 - 2^{-5-b}$, being b a constant. In order to accelerate training, the authors also propose a chunk-wise recurrent paradigm inspired by the aforementioned inter/intra-level recurrent approaches. *Inter-chunk recurrence* propagates the hidden states at chunk-level, followed by *intra-chunk parallel computation* that directly computes the output Y based on the chunk-level hidden states. This approach allows to parallelize computations within a chunk without explicitly *materializing* the intermediate hidden states in the high bandwidth memory (HBM) of the GPU (Dao et al., 2022) (see Section 8.2 for further details on hardware efficiency). Formally, the input U is split into non-overlapping chunks, where each chunk is of length C. Let $S_{[i]} \in \mathbb{R}^{d_k \times d_k}$ be the chunk-level hidden state after processing i chunks, i.e., $S_{[i]} := S_{iC}$. The query vector corresponding to the i-th chunk is defined as $Q_{[i]} := Q_{iC+1:(i+1)C} \in \mathbb{R}^{C \times d_k}$, and $K_{[i]}$, $V_{[i]}$, $O_{[i]}$ are similarly defined. Then, for $i \in [0, 1, \dots \frac{L}{C} - 1]$, the inter-chunk recurrence is defined as,

$$S_{[i+1]} = \gamma^C S_{[i]} + K'_{[i+1]}(V_{[i+1]} \odot \Gamma),$$

where $\Gamma_{ij} = \gamma^{C-i}$ for all j. The sum of all RNN inputs from a chunk (i.e., $K'_{[i]} V_{[i]}$) can be computed *before* the recurrence in parallel in $\mathcal{O}(C^2 d_{\text{in}})$. The intra-chunk parallel computation is given by,

$$Y_{[i]} = \underbrace{\left(Q_{[i]} S_{[i-1]}\right) \odot \zeta}_{\text{cross-chunk}} + \underbrace{\left(Q_{[i]} K'_{[i]} \odot D\right) V_{[i]}}_{\text{intra-chunk}},$$

$$\zeta_{ij} = \gamma^{i+1}, \ \forall j.$$

The *intra-chunk* component is a linear attention performed on the chunk, and thus takes $\mathcal{O}(C^2 d_k + C {d_k}^2)$, while the *cross-chunk* integrates the previous chunk component for the contribution from the hidden state from the previous chunk, and takes $\mathcal{O}(C {d_k}^2)$. Overall, training complexity is $\mathcal{O}\left(\frac{L}{C}(C^2 d_k + C {d_k}^2)\right) = \mathcal{O}(LC d_k + L {d_k}^2)$. The chunk size C can be controlled for a trade-off between FLOPs and wall-clock speed. Overall, the decay factor introduced by RetNet puts more weight on recently processed inputs and is independent on the processed data. Finally, RetNet exploits multiple heads equipped with retention and a different γ for each head, resulting in different variance statistics. GroupNorm (Wu & He, 2018) normalizes the output of each head, and a swish gate (Hendrycks & Gimpel, 2016) increases the non-linearity of retention layers. RetNet is not the only approach belonging to this category. TransNormerLLM (Qin et al., 2023a) is claimed to be the first linear attention-based Large Language Model (LLM) (up to 175 billion parameters) that outperforms conventional softmax attention-based models in terms of both accuracy and efficiency. It builds upon TransNormer (Qin et al., 2022a) (see Section 3.1), replacing diagonal attention with linear attention. addresses the issue of attention dilution by adding linearized relative positional encodings with exponential decay (Qin et al., 2023b), linear attention acceleration (by leveraging the recomputation technique from FlashAttention (Dao et al., 2022) to avoid the materialization of the 2D hidden state S_t – see Section 8.2), tensor normalization from Qin et al. (2022a), and a gating mechanism with a decay factor applied to the additive recurrent update. GateLoop (Katsch, 2023) incorporates a data-controlled gating mechanism which is applied on inputs, hidden states and outputs, replacing the fixed decay rate exploited in RetNet with a time-varying data-dependant diagonal state transition $A_t \in \mathbb{C}^{d_s \times d_s}$, defined in polar form,

$$\begin{aligned} A_t &= \text{diag}(\gamma_t e^{i\theta_t}) := \text{diag}(\sigma(\alpha_t) e^{i\beta_t}), \\ X_t &= A_t S_{t-1} + k_t \otimes v_t, \\ y_t &= S_t q_t, \end{aligned} \tag{31}$$

where $S_t \in \mathbb{C}^{d_s \times d_s}$, α_t, β_t are learned linear projections of the input x_t and σ is the sigmoid activation. Indeed, similarly to RetNet and other recent works (i.e., LRU(Orvieto et al., 2023), see Section 4), the magnitude and phase of the state transition A_t are controlled separately. Interestingly, the authors remark how q_t and k_t act as input and output

gates, respectively, and A_t can be interpreted as a forget/retain gate on the linear recurrence. Unfolding the recurrence in Eq. (31) yields $y_t = \sum_{m=1}^{t} q_t'(k_m \otimes v_m) \prod_{j=m+1}^{t} A_j$, that equals the RetNet output computation (Eq. (29)) if we fix the state transition gate, $y_t = \sum_{m=1}^{t} q_t'(k_m \otimes v_m) A^{t-m}$. Additionally, GateLoop leverages an efficient associative scan computation for efficient parallelized computation of the linear recurrence (see Section 4). A concurrent work, ReLiT (Pramanik et al., 2023), investigates index-wise outer-product based gating functions instead of the scalar one available in previous works (Irie et al., 2022b; Schlag et al., 2021) (e.g., g_t and β_t), as long as an approximation based on trigonometric functions, referred to as AReLiT. The model is a kernel-based Linear Transformer where the authors propose learnable feature maps ϕ instead of fixed ones. Gated Linear Attention (GLA)(Yang et al., 2023) explores a data-dependent[12] gating mechanism for linear Transformers, and propose both parallel and block-parallel forms that can take advantage of tensor core computations on modern accelerators (GPUs, TPUs). The recurrent form updates the recurrent state S_t by computing a gating matrix produced by means of an outer-product (similar to Mao (2022) and Pramanik et al. (2023)), i.e., $G_t = \alpha_t \otimes \beta_t \in \mathbb{R}^{d_k \times d_k}$, where $\alpha_t, \beta_t \in \mathbb{R}^{d_k}$. A possible instance of this form is the following one, where α_t is a low-rank re-parametrization of the input and β_t is a column vector filled with ones,

$$\begin{aligned} \alpha_t &= \sigma\left(W_\alpha^2 W_\alpha^1 u_t + b_\alpha\right)^\tau, \\ \beta_t &= \mathbb{1}, \\ S_t &= G_t \odot S_{t-1} + k_t \otimes v_t, \end{aligned} \tag{32}$$

where σ is the sigmoid function, $W_\alpha^1 \in \mathbb{R}^{16 \times d_k}$, $W_\alpha^2 \in \mathbb{R}^{d_k \times 16}$ implement a low-rank parametrization, $\tau \in \mathbb{R}$ is a temperature term to encourage the model to have a slower forgetting rate. Overall, the output of the recurrent form of the GLA layer is,

$$\begin{aligned} o_t &= S_t q_t, \\ r_t &= \text{Swish}(W_r u_t + b_r), \\ y_t &= (r_t \odot \text{LayerNorm}(o_t)) W_o, \end{aligned} \tag{33}$$

where the LayerNorm after o_t follows prior work (also referred to as NormAttention) Qin et al. (2022a, 2023a), Sun et al. (2023a). The final output y_t is obtained by following the structure of a GAU layer from the FLASH model (Hua et al., 2022), where an additional output gate r_t with the Swish activation (Hendrycks & Gimpel, 2016) is used. The GLA parallel form is computed as follows,

$$O = \left(\left((Q \odot B) \left(\frac{K}{B} \right)' \right) \odot M \right) V,$$

where $B \in (0,1)^{L \times d_k}$ is the matrix obtained by stacking $b_t := \prod_{j=1}^{t} \alpha_j$ and M denoted the causal mask.[13] GLA also provide a parallel and two-level chunk-wise block-parallel forms. See Section 8.2 for further details on their hardware-aware solutions.

This section shows that the once-sharp boundary between self-attention and recurrence is rapidly eroding. Modern Transformer research first adopts kernel or linear attention schemes (e.g., Linear Transformer, Performer) that rewrite the softmax dot product as a feature-kernel product. This reframes attention as an additive update of a constant-size matrix state and cuts the quadratic $\mathcal{O}(L^2)$ cost of vanilla attention to linear $\mathcal{O}(L)$ while preserving full parallel training. Extending that idea, fast-weight and delta-rule methods such as DeltaNet and SRWM treat the key-value cache as a programmable associative memory, using outer-product updates and data-controlled forgetting gates to balance capacity and recency. A parallel line of work eliminates dot-products entirely: low-rank or channel-wise interactions in Attention-Free Transformers and the RWKV family compute gated exponential averages that naturally down-weight distant tokens and enable linear-time decoding. Finally, segment and chunk recurrent architectures such as Transformer-XL, RetNet and their variants propagate compressed hidden states across windows, extending effective context length without incurring quadratic growth. Together these threads reveal a unifying trend: contemporary Transformers increasingly resemble specialised recurrent networks, compressing history into compact learned states, applying explicit decay or memory tokens to regulate long-range influence, and thereby reclaiming much of the efficiency once unique to classic RNNs.

While this section demonstrated that many Transformer variants now reuse recurrent ideas, an orthogonal line of work pursues the same goal from first principles: *State-Space Models* (SSMs). By discretising continuous-time dynamics, SSMs encode sequence history in compact linear states and thus inherit the favourable $\mathcal{O}(L)$ scaling of classical RNNs without relying on attention. Section 4 reviews this rapidly growing family and its role in long-sequence processing.

4. Deep state-space models

Recent works on models that are intrinsically based on recurrent computations particularly emphasize the notion of (deep) State-space Models (Gu et al., 2021b). In particular, there exists a growing interest in exploiting the computational advantages of using multiple stacked instances of linear recurrences, whose dynamics are appropriately conditioned to avoid trivial explosive/vanishing dynamics. The development of such lines of works can be traced back to the seminal work of Voelker et al. (2019) and Gu et al. (2020a), in which the authors propose methods to perform *online function approximation*. Then, the scientific literature of the last years transitioned from focusing on online function approximation to specifically designed (deep) State-Space Models (Gu et al., 2021a,b) and more advanced architectures exploiting them De et al. (2024), Gu and Dao (2023), as we anticipate in Fig. 15. Table 4 provides an overview of the complexities of recent Deep State-Space Model variants, which we discuss in the following sections.

Online Function Approximation The basics of online function approximation, with regard to the first works on this novel wave of state-space models, can be formalized as follows. Given a function of one variable defined on the half line $u : [0, +\infty) \to \mathbb{R}$, the problem of online approximation of such function is twofold: (i) for time instant $t \in [0, +\infty)$ find an approximation of u until t, i.e., $u^t := u|_{I_t}$ with $I_t := (0, t)$ and (ii) have a method to update online such approximation. In order to formalize the concept of approximation, we need to have some notion of closeness, and hence we assume that the function we want to approximate lies in some normed space. Moreover, the measure with respect to which we define the notion of integrability plays a rather important role in computing an online approximation. In the following descriptions, we will mostly refer to the seminal works in Gu et al. (2020a), where the HiPPO model/theory is introduced, and Voelker et al. (2019), based on Legendre Memory Units (LMUs). In Gu et al. (2020a) the authors find that working with a normalized Lebesgue measure on I_t (which is the

Table 4
Complexities for sequences of length L, comparing representatives of the categories of Deep State-Space models discussed in Section 4, vanilla Transformers and Recurrent Models (Section 2). In order to bridge the discussion in the main text of paper, here we assumed $d_k = d_{\text{in}}$, for the sake of simplicity.

	Model	Recur.	Time	Space
Section 2	Rec. Net	Yes	$\mathcal{O}(Ld_{\text{in}}^2)$	$\mathcal{O}(Ld_{\text{in}})$
Section 3	Transf.	No	$\mathcal{O}(L^2 d_{\text{in}})$	$\mathcal{O}(L^2 + Ld_{\text{in}})$
Section 3.1	H3	Yes	$\mathcal{O}(Ld_{\text{in}}(\log L + d_{\text{in}}))$	$\mathcal{O}(Ld_{\text{in}})$
Section 3.2	S4	Yes	$\mathcal{O}(Ld_{\text{in}}^2)$	$\mathcal{O}(Ld_{\text{in}})$
Section 3.3	Hyena	Yes	$\mathcal{O}(Ld_{\text{in}}(\log L + d_{\text{in}}))$	$\mathcal{O}(L(\log L \cdot d_{\text{in}}))$
Section 3.4	Mamba	Yes	$\mathcal{O}(Ld_{\text{in}})$	$\mathcal{O}(d_{\text{in}})$

[12] This differs, for instance, from the gating mechanism implemented by RetNet, which decays over time independently with respect to the input.

[13] For stability, it is computed in log space (see the referenced paper for further details).

standard choice in $\mathbb{R}^n$) has several advantages. A different choice is explored in LMU (Voelker et al., 2019) that, in light of the theoretical formulation of the problem presented in Gu et al. (2020a), correspond to choosing a measure of density that is constant on a window $[t-\theta, t]$ of size θ just before the end point t of the considered temporal instant. The other basic ingredient to consider in function approximation is the class of basis functions with which we want to perform such an approximation. In Gu et al. (2020a) the authors consider the case of translated and rescaled Legendre polynomials, v_n^t for $n = 1, 2, \ldots$, defined in $[0, t]$ by,

$$v_n^t(x) = \sqrt{2} e_n\left(\frac{2x}{t} - 1\right) \quad \forall x \in [0, t], \quad n = 0, 1, \ldots, \tag{34}$$

where e_n are normalized Legendre polynomials (see Ciarlet (2013)). A similar choice has been also done in Voelker et al. (2019). Then, the wanted approximation v^t of the function u^t can be expressed (as explained in Gu et al. (2020a)) by,

$$v^t = \sum_{n=0}^{N-1} c_n(t) v_n^t \quad \text{where} \quad c_n(t) := (u^t, v_n^t)_t, \tag{35}$$

where $(u^t, v_n^t)_t := \int_{I_t} u^t v_n^t \, dx/t$ is the standard scalar product in $L^2((0,t);\mathbb{R})$ rescaled by a factor $1/t$. More precisely, since the goal is to solve an approximation problem on I_t, in order to define integrability we will consider the Lebesgue measure $\mathcal{L}^1$ on $\mathbb{R}$ restricted to I_t and we will define $\forall t > 0$ the rescaled measure $\mathcal{L}_t^1$ such that $\mathcal{L}_t^1(A) = \mathcal{L}^1(A)/t$ for all $A \subset I_t$.[14] One we have this measure we can define the Hilbert space $L^2_{\mathcal{L}_t^1}(I_t;\mathbb{R})$ which is exactly the space of square $\mathcal{L}_t^1$-integrable, real-valued functions. So it is natural to require that the method that we will develop works on functions $u : \mathbb{R}_+ \to \mathbb{R}$ such that for all $t > 0$ $u|_{I_t} \in L^2_{\mathcal{L}_t^1}(I_t;\mathbb{R})$. The approximation problem then can be stated as the problem of finding a solution to the following minimization problem,[15]

$$\min_{v \in V_N^t} \|v - u^t\|_{L^2_{\mathcal{L}_t^1}(I_t;\mathbb{R})}, \tag{36}$$

where $V_N^t \subset L^2_{\mathcal{L}_t^1}(I_t;\mathbb{R})$ is a finite, N-dimensional subspace that we assume to be spanned by N *orthonormal* basis functions $v_0^t, \ldots, v_{N-1}^t$; i.e., $V_N^t := \operatorname{span}\{v_0^t, \ldots, v_{N-1}^t\}$. Here othornormality as usual means that $(v_i^t, v_j^t)_t = \delta_{ij}$ for all $i, j = 0, \ldots, N-1$ where δ_{ij} is the usual Kronecker delta and $(\cdot,\cdot)_t$ is the standard scalar product in $L^2_{\mathcal{L}_t^1}(I_t;\mathbb{R})$, that is $(f,g)_t := \int_{I_t} f g \, d\mathcal{L}_t^1 \equiv (\int_{I_t} f g \, dx)/t$ being dx the usual notation for the Lebesgue measure. In general the solution to the problem in Eq. (36) (see Ciarlet (2013)) is given by

$$v^t = \sum_{n=0}^{N-1} c_n(t) v_n^t \quad \text{where} \quad c_n(t) := (u^t, v_n^t)_t. \tag{37}$$

The crucial result presented in Gu et al. (2020a) and Voelker et al. (2019) is that the computation of the coefficients c_n defined above can be done using system of ordinary differential equations with a Cauchy initialization so that they can be estimated in an online manner. In particular if we denote with $c := (c_0, \ldots, c_{N-1})$ then c can be computed as a solution of

$$\dot{c}(t) = A(t)c(t) + B(t)u(t), \tag{38}$$

where the matrix $A(t)$ and the vector $B(t)$ can be explicitly computed. In particular in the HiPPO setting these matrices turns out to be (see Appendix E of Gu et al. (2020a)),

$$A_{ij}(t) = -\frac{1}{t}\begin{cases} \sqrt{(1+2i)(1+2j)} & \text{for } i > j \\ 1+i & \text{for } i = j \\ 0 & \text{for } i < j \end{cases}, \qquad B_i(t) = \frac{1}{t}\sqrt{1+2i}, \tag{39}$$

where the temporal dependence takes the form of a rescaling $1/t$ that, in turn, comes from the choice of the measure $\mathcal{L}_t^1$ defined above.

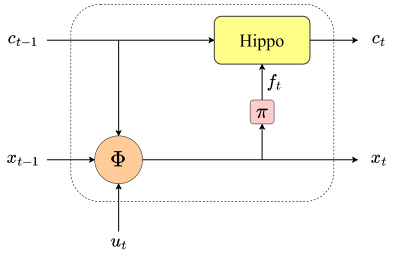

Fig. 12. Online function approximation methods based in ODE, such as HiPPO (Gu et al., 2020a), can be seamlessly added to a recurrent computation of a standard RNN. We indicate with Φ the state transition function, π is the "one dimensional" projection described in the main text of the paper, and the HiPPO module performs the integration of the HiPPO equations, Eq. (38), with u replaced by f.

From Online Approximation to Deep State-Space Models Online function approximation *is not* a learning problem; however the results of Voelker et al. (2019) and Gu et al. (2020a) discussed above show that the coefficients of the online approximation can be efficiently updated using a linear recurrence relation. Hence, in both works the authors propose a direct application of this idea to learning, using this online approximation mechanism inside a recurrent network to maintain a compact representation of a projection of the state over the whole past times (Fig. 12). More precisely, the state of the RNN at time t is updated using the following update rule,

$$x_t = \Phi(x_{t-1}, c_{t-1}, u_t), \tag{40}$$

where Φ is the transition function that depends on the precise recurrent architecture, u_t is the input of the net at time t and c_{t-1} are the coefficients of the online approximation of the function $f_t = \pi(x_t)$, where $\pi : \mathbb{R}^{d_s} \to \mathbb{R}$ is a projection of the state onto the real line (which is necessary, since these online approximation methods work on scalar functions). The leap from this hybrid model, where the state of the recurrence is enriched with an online approximation of the state itself, to Deep State-Space Models has been proposed in Gu et al. (2021b), which analyzes Linear State Space Layers (LSSLs) in comparison to other deep learning models that are used to process sequences, and in Gu et al. (2021a), which refines such models to address computational limitations of the former model. In Linear State Space Layers (Fig. 13), the main idea is to use a linear continuous-time expression to model the update of the state itself,

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{41}$$

where the input signal $t \mapsto u(t) \in \mathbb{R}$ is one dimensional, and processing of higher dimensional signals of features is achieved by learning independent models for each input dimension. The matrix $A \in \mathbb{R}^{d_s \times d_s}$ while $B \in \mathbb{R}^{d_s \times 1}$. As it is customary in state space models, the output trajectory of the model, that we will denote as $t \mapsto y(t) \in \mathbb{R}$, is then computed via another "static" (i.e., not involved in a recurrence) linear map,

$$y(t) = Cx(t) + Du(t), \tag{42}$$

where $C \in \mathbb{R}^{1 \times d_s}$ and $D \in \mathbb{R}$.

The continuous time model described by Eq. (41) is typically discretized in order to be numerically implemented. Different discretiza-

[14] Notice that $\mathcal{L}_t^1$ is a probability measure on I_t since beside being a well defined measure we also have that $\mathcal{L}_t^1(I_t) = 1$.

[15] This problem has always a unique solution since the subspace V_N^t is finite dimensional and hence it is closed

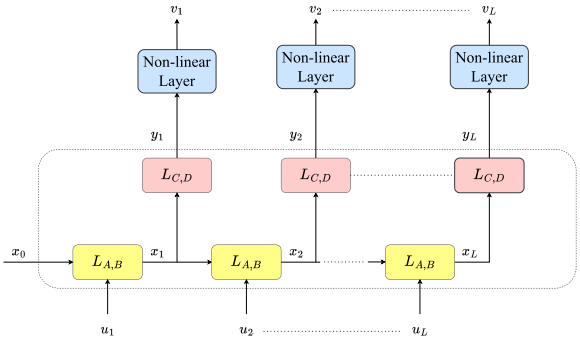

Fig. 13. A Linear State Space Layer, LSSL, is a *nonlinear* transformation that maps a sequence of length L, $(u_1, \dots, u_L)$ into another sequence $(v_1, \dots, v_L)$ through a *linear* recurrence and a non-linear feed-forward layer. In this figure we indicate with $L_{A,B}$ the linear transformation that defines the transition function of the recurrence with parameters A and B (as in Eq. (41)), while $L_{C,D}$ is the linear function that define the static mapping $x \mapsto Cx + Du$.

tion techniques can be applied, the more direct being the explicit (or forward) Euler method,[16]

$$x_{t+1} = x_t + \tau\big(Ax_k + Bu_{t+1}\big). \tag{43}$$

Where, (x_t) and (u_t) are sequences. More generally, a typical discrete approximation of Eq. (41) will have the form

$$x_{t+1} = A^\tau x_t + B^\tau u_{t+1}, \tag{44}$$

where for instance $A^\tau = \mathrm{I} + \tau\mathrm{A}$ and $B^\tau = \tau B$ for the forward Euler scheme described in Eq. (43), being I the identity matrix. Another very common discretization scheme for Eq. (41) used in the context of Deep State-Space Models (see Gu et al. (2021b)) is the *bilinear method*, that is equivalent to the choice $A^\tau = (\mathrm{I} - (\tau/2)\mathrm{A})^{-1}(\mathrm{I} + (\tau/2)\mathrm{A})$ and $B^\tau = (\mathrm{I} - (\tau/2)\mathrm{A})^{-1}\tau\mathrm{B}$. On the other hand, the output map described in Eq. (42) remains exactly the same and defines, in the discrete setting, the sequence of outputs $(y_t)_{t>0}$ defined in terms of the state sequence as $y_t = Cx_t + Du_t$. Assuming for definiteness that $x_t \equiv 0$ if $t < 0$, the recursion relation in Eq. (44) can be unfolded to obtain a closed expression for the t-th element of the sequence of the state in terms of the inputs $u_0, \dots, u_t$. Indeed, it is immediate (by repeated use of Eq. (44)) that,

$$\begin{aligned} y_t &= CB^\tau u_t + CA^\tau B^\tau u_{t-1} + C(A^\tau)^2 B^\tau u_{t-2} \\ &\quad + \cdots + C(A^\tau)^t B^\tau u_0 + Du_t \\ &= \sum_{j=0}^{t} C(A^\tau)^j B^\tau u_{t-j} + Du_t. \end{aligned}$$

Now, if we define the sequence of real numbers $(p_t^\tau)_{t\geq 0}$ as $p_t^\tau := C(A^\tau)^t B^\tau \in \mathbb{R}$, the outputs can be expressed as a convolution of the input u with the sequence $(p_t^\tau)_{t\geq 0}$,

$$y_t = \sum_{j=0}^{t} p_j^\tau u_{t-j} + Du_t. \tag{45}$$

This is what is commonly referred to as the convolutional form of the linear state space model in Eq. (44) – see Fig. 14-top. Now, going back to LSSL (Gu et al., 2021b), the matrix A is represented with a suitable matrix factorization, i.e., $A = P(D + T^{-1}Q)$, with D, P and Q diagonal matrices and T tridiagonal. The HiPPO matrix $A(t)$ defined in Eq. (39) admits such factorization (see Appendix E.2 of Gu et al. (2021b)). In this way, matrix A is guaranteed to be quasiseparable, a property that is presented as desirable both for handling long dependencies and for enabling efficient matrix-vector multiplication. As a common practice in deep learning, several LSSLs can be stacked together, each layer receiving as input the output of the previous one. This is possible since

[16] Technically this is a mixed scheme since the input is computed at the step $t + 1$, however we call it explicit since it is so with respect to the state variable.

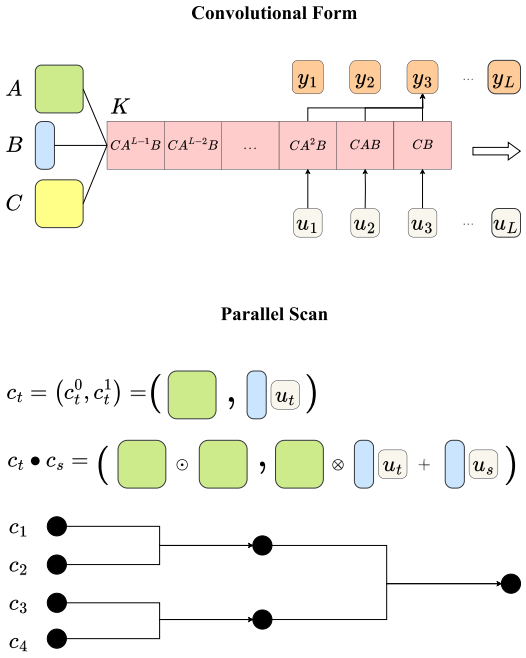

Fig. 14. Methods to parallelize computations over sequence length in Linear RNNs: Convolutional form, computed with FFT transform (Gu et al., 2021a), and Associative Parallel Scans (Martin & Cundy, 2018; Smith et al., 2022).

the input and output have the same dimensionality. The main problem with this architecture is the computational cost; indeed (see Gu et al. (2021a)), it requires $\mathcal{O}(d_s{}^2 L)$ operations and it scales as $\mathcal{O}(d_s L)$ for what concerns memory in order to compute the input-output mapping in Eq. (44). In order to overcome this precise limitation the S4 model (Gu et al., 2021a) has been introduced to condition the structure of the matrix A. There is a large set of works that were published in the last few years along this line of research, and that, starting from S4, we describe in the following. Refer to Fig. 15 for an overview.

S4 The Structured State Space Sequence Model (S4) (Gu et al., 2021a) is based on the continuous-time linear system in Eqs. (41) and (42). The matrix A is imposed to have the following form,

$$A = \mathrm{diag}(\lambda_1, \dots, \lambda_{d_s}) + PQ^\dagger, \tag{46}$$

where $\mathrm{diag}(\lambda_1, \dots, \lambda_{d_s})$ is a diagonal matrix, $\lambda_i \in \mathbb{C}$ for every i and $P, Q \in \mathbb{C}^{d_s \times 1}$. In Eq. (46), the $\dagger$ operation denotes the conjugate transpose and the term $PQ^\dagger$ is usually referred to as *low-rank correction*. With this particular choice for A, the computation of the recursion in Eq. (41) has complexity $\tilde{\mathcal{O}}(d_s + L)$. For discretizing the dynamics in Eqs. (41) and (42), the bilinear transform with discretization step τ is applied, leading to the already introduced,

$$\begin{aligned} A^\tau &= \Big(I - \frac{\tau}{2}A\Big)^{-1}\Big(I + \frac{\tau}{2}A\Big), \\ B^\tau &= \Big(I - \frac{\tau}{2}A\Big)^{-1}\tau B. \end{aligned} \tag{47}$$

The model follows a single input single output (SISO) structure, meaning each component of the input (called *input channel*) u_i, for $i = 1, \dots, d_{\text{in}}$, is processed by a distinct discretized system, each generating a scalar output y_j, for $j = 1, \dots, d_{\text{in}}$ (notice that $d_y = d_{\text{in}}$). The dynamics matrix A for each of the d_{in} SISO subsystems is initialized according to HiPPO theory. While the original S4 does not inherently favor initialization towards marginal stability to maintain long-range memory, the subsequent work SaShiMi (Goel et al., 2022) ensures stability by enforcing the real part of λ_i to be negative, $\mathrm{Re}(\lambda_i) \in \mathbb{R}^-$, for every i. For training S4, the convolutional representation in Eq. (45) of the output is used and the structure of A^τ in Eq. (46) is exploited for efficiently computing

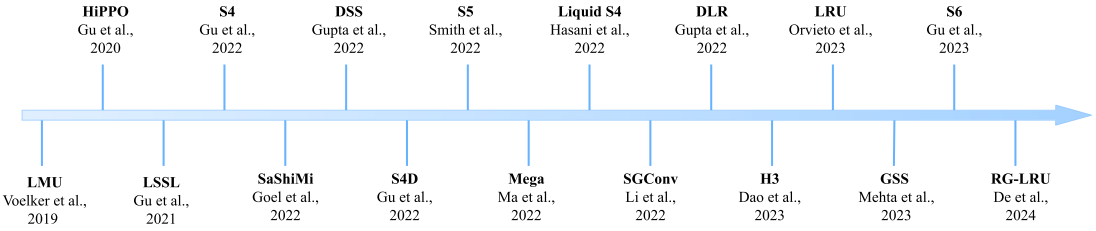

Fig. 15. Timeline showing the chronological development of deep state-space models—Overview of the organization of Section 4.

its inverse. At inference time, a recurrent representation of the model,

$$\begin{aligned} x_{t+1} &= A^\tau x_t + B^\tau u_t, \\ y_t &= C x_t + D u_t, \end{aligned} \tag{48}$$

is directly used. Subsequent works show that it is possible to match the performances of S4 even without the low rank correction, but still retaining the initialization of the diagonal part to be consistent with the diagonal part of the HiPPO matrix. The diagonal structure of the matrix A leads to the *Diagonal State Space* (DSS) model (Gupta et al., 2022a) and this work is theoretically expanded in the infinite width setting in Gu et al. (2022), leading to S4D.

S4D The Diagonal Structured State Space Sequence Model (S4D) (Gu et al., 2022) builds upon S4, and it assumes that the matrix A has a diagonal structure,

$$A = \mathrm{diag}(\lambda_1, \dots, \lambda_{d_s}), \tag{49}$$

which yields computational improvements. In order to get its discrete-time version, exact discretization is applied to the dynamics of Eqs. (41) and (42), with discretization step τ, leading to,

$$\begin{aligned} A^\tau &= e^{\tau A}, \\ B^\tau &= (\tau A)^{-1}(A^\tau - I)\tau B. \end{aligned} \tag{50}$$

S4D retains the SISO structure from S4 and its initialization is still based on HiPPO theory. Similar to SaShiMi, the eigenvalues of A used for initialization lie in $\mathbb{R}^-$. Again, convolutional representation of Eq. (45) is used in training, and the recurrent representation in Eq. (48) is used during inference. The diagonal structure of the matrix A^τ allows for efficient computation of the discretization in Eq. (50). The SSMs described so far, as showed in Dao et al. (2023), struggle with tasks like recalling earlier tokens and comparing tokens across a sequence when applied to language modeling tasks (see also Section 8.1). The H3 model is explicitly designed to tackle these challenges and will be described in the following, right after having extended the SISO family of models to the more advanced MIMO.

From SISO to MIMO: S5 The Simplified Structured State Space Sequence Model (S5) (Smith et al., 2022) is the first Deep SSM to be parameterized leveraging the concept of multiple input multiple output (MIMO) systems, for simplifying the architectural components and enhancing computations. This means that the full input vector $u \in \mathbb{R}^{d_{\text{in}}}$ is fed into a single bigger MIMO system of Eq. (48), instead of d_{in} SISO scalar smaller subsystems, by stacking the matrices A^τ, B^τ, C, used in S4 and S4D. S5 inherits the S4D parametrization of the matrix A (i.e., it is a diagonal matrix), while it can be discretized applying both bilinear (as done in S4, see Eq. (47)) and exact (as done in S4D, see Eq. (50)) discretizations. The MIMO structure and the diagonal parameterization allows for parallel computation of the individual output components via a *parallel scan* algorithm (see Appendix H of Smith et al. (2022)). The parallel scan algorithm (see Fig. 14-bottom) offers a way to parallelize the computations of a sequence of elements of a *semigroup* $(S, \bullet)$ generated by a recurrence relation of the form $s_{i+1} = s_i \bullet c_i$, where $(c_i)_{i=1}^L$ is a given sequence of elements of S.[17] This approach can be directly applied to the computation of a linear recurrence of the form in Eq. (44) with the following choices,

1. $S = \{(M, v) : M \in \mathbb{R}^{d_s \times d_s} \quad \text{and} \quad v \in \mathbb{R}^{d_s}\}$;
2. $(M, v) \bullet (N, u) := (NM, Nv + u)$ for all $(M, v) \in S$ and $(N, u) \in S$;
3. $c_i = (A^\tau, B^\tau u_i)$.

Indeed one can show (see Appendix H of Smith et al. (2022)) that the sequence

$$\begin{cases} s_0 = (\mathrm{I}, 0) \in \mathrm{S} \\ s_{i+1} = s_i \bullet (A^\tau, B^\tau u_i) \end{cases}$$

has the following representation in terms of the solution x_k of Eq. (44) with zero initialization $x_k \equiv 0$ for $k \le 0$,

$$s_k = ((A^\tau)^{k-1}, x_k) \quad k \ge 0.$$

Therefore, computations at training and inference time are made efficiently in the recurrent representation of Eq. (48). HiPPO theory is again used for initializing the matrix A, obtaining the same starting eigenvalues of S4D. Together with S5, some novel variants of S4 are introduced. Recent literature describes also Mega (Ma et al., 2023) (see Section 3) as a SSM. Indeed, it can be interpreted as a simplification of S4 to a diagonal SSM where the values in the diagonal of the matrix A are restricted to be real numbers, interpreting it as an exponential moving average (EMA). Liquid S4 (Hasani et al., 2021) exploits the original S4 formulation (with low-rank correction) combined with liquid time-constant networks (please refer to Section 5.1 for further details on liquid time-constant networks). SGConv model (Li et al., 2022b) leverages the convolutional form of Eq. (45) to obtain a filter-based version of S4. Up to this point, SSMs rely on discretizing the continuous time dynamics in Eq. (41). The authors of Gupta et al. (2022b) eliminate the discretization step and introduce a model based on vanilla *Diagonal Linear RNNs* (DLR), closely related to DSS and S4D, in which each input is processed independently at each layer. Here, the discretization step is directly absorbed into the continuous-time transition matrix A. The authors show that, after numerical integration, diagonal state-space models and linear RNNs share the same function approximation class.

SSMs in Language Modeling: H3 The Hungry Hungry Hippo (H3) (Dao et al., 2023) model is a novel approach to leverage SSMs for language modeling, aiming to address the limitations of previous SSMs in tasks like Associative Recall and Induction Heads (see Section 7) compared to attention-based models. H3 draws inspiration from linear attention, which assumes a specific form for the similarity metric used in attention calculations. It stacks two discrete SSMs: one with a shift matrix (i.e., a local convolution) to remember past tokens and one with a diagonal matrix to retain state over the entire sequence. The key innovation lies in introducing *gates* (i.e., multiplicative interactions) between the outputs of these SSMs and projections of the input . Combined with the shift matrix, these gates enable H3 to compare tokens across the sequence. The shift SSM identifies specific events (like the presence of a key token), while the diagonal SSM stores and retrieves associated information (like the corresponding value token). H3 has a time complexity of $\mathcal{O}(L \log(L))$ for a sequence of length L, making it asymptotically more efficient than traditional attention, which has a complexity of $\mathcal{O}(L^2)$.

[17] We recall that a semigroup consist of a set S together with an associative operation $\bullet$.

LRU Another model belonging to the family of MIMO systems, as S5, is the Linear Recurrent Unit (LRU) (Orvieto et al., 2023). The pre-processing of the input and post-processing of the output are also identical to those in S5. Instead, LRU is the first of the SSMs that does not come from a discretization of a continuous-time model, since a discrete parametrization of A^τ and B^τ is directly used. Indeed, it parameterizes the discrete-time dynamics in Eq. (48) as,

$$A^\tau = e^{-e^{diag\left(\lambda_1,\dots,\lambda_{d_s}\right)}+i\,diag\left(\theta_1,\dots,\theta_{d_s}\right)}, \qquad B^\tau = e^\gamma \Gamma, \tag{51}$$

where i is the complex unit, $\lambda_j, \theta_j \in \mathbb{R}$ for every $j = 1, \dots, d_s$, $\Gamma \in \mathbb{C}^{d_s \times d_{\text{in}}}$ is a dense complex-valued matrix, and $\gamma \in \mathbb{R}$. Given this parameterization, the eigenvalues of A^τ in polar coordinates (i.e., $a_j = r_j + i\,\theta_j$ where $r_j = e^{-e^{\lambda_j}}$) are constrained to lie in the unit-disk, by construction. The initialization is then directly performed in polar coordinates by defining a range for r and θ in which they are uniformly sampled. This provides an alternative to the HiPPO theory for initialization (the HiPPO theory is instead used in S4, S4D and S5). Moreover, it is also the first formalization where A^τ and B^τ do not share parameters. As for S5, the model is implemented using a parallel scan algorithm for both training and inference.

S6: Mamba Unlike previous models, the Scan Selective Structured State Space Sequence Model (S6) (Gu & Dao, 2023) introduces a linear time-varying representation of the dynamics in Eqs. (41) and (42), which is referred to as a "selection mechanism". This is achieved by letting the parameters that affect interactions along the sequence (e.g., the recurrent dynamics of the RNN) be input-dependent. Indeed, the matrices B and C are now functions of the input u_t, at every time-step t, parametrized as,

$$\begin{aligned} B_t &= W_B u_t \\ C_t &= W_C u_t, \end{aligned} \tag{52}$$

where W_B and W_C are linear projection matrices of appropriate dimensions. Similar to S4D, the matrix A is a time-invariant diagonal matrix, as in Eq. (49), and it uses exact discretization to compute the discrete-time dynamics of Eq. (48). However, in S6, also τ is time-varying and function of the input, leading to the discretization,

$$\begin{aligned} \tau_t &= \text{softplus}(W_\tau u_t), \\ A_t^\tau &= e^{\tau_t A}, \\ B_t^\tau &= (\tau_t A)^{-1}(A_t^\tau - I)\tau_t B_t, \end{aligned} \tag{53}$$

where $C_t^\tau = C_t$, $D_t^\tau = D_t$ and $W_\tau \in \mathbb{R}^{1 \times d_{\text{in}}}$. The model is structured in a MIMO manner (as S5, LRU and S6) and the dynamic matrix A is initialized with $\lambda_j = -j$, for every $j = 1, \dots, d_s$, ensuring that the eigenvalues lie in the negative halfplane. Since τ_t is time-varying, the eigenvalues of A_t^τ have an initialization that is input-dependent. The time-varying representation presents computational challenges, despite being more expressive. The authors provide an efficient implementation of the time-varying dynamics in Eq. (48), presenting a variation of the parallel scan and exploiting it both at inference and training time. S6 introduces an innovative way of pre-processing the input, called Mamba, which relies on both linear and non-linear maps. The input enters the recurrence through a linear projection, followed by a causal convolution. Additionally, it passes through a linear projection followed by a SiLU nonlinearity before entering the gating function for post-processing. The gating function is inspired by previous models, i.e., H3 and GAU. An architecture close to Mamba is the *Gated State Space* (GSS) layer (Mehta et al., 2022), again inspired by GAU. GSS resembles Mamba but includes additional projections. The key difference is that GSS's projection reduces the model dimension to decrease the state size of the SSM, whereas Mamba expands the model dimension to increase the state size.

RG-LRU: Hawk and Griffin The Real-Gated Linear Recurrent Unit (RG-LRU) (De et al., 2024) fuses ideas from LSTMs, LRU, and S6. As in LRU, the RG-LRU model is structured by means of a MIMO system and, as in S6, the parametrization of the linear dynamics is time-varying. Unlike all previous SSMs, the matrices C and D are not present here. RG-LRU does not rely on a continuous-time representation (the same thing happens in LRU) and directly parametrizes the discrete matrices A_t^τ, B_t^τ as,

$$\begin{aligned} A_t^\tau &= e^{-c \cdot \text{softplus}(W_A)\sigma(W_\tau u_t)}, \\ B_t^\tau &= \sqrt{1 - A_t^2}\,\sigma(W_B u_t), \end{aligned} \tag{54}$$

where $\sigma(\cdot)$ is the sigmoid function, W_τ, W_A, W_B are linear projection matrices (of appropriate dimensions) initialized with standard initialization methods, e.g., Glorot, and $c \in \mathbb{R}$ is a scalar constant. The square root operation is computed element-wise. Given this parameterization of A_t^τ, its eigenvalues are restricted to the unit disk by construction. The implementation of RG-LRU assumes that the state and input dimensions coincide, i.e., $d_s = d_{\text{in}}$. Since the parametrization in Eq. (54) is time varying, RG-LRU exploits a customized variation of the parallel scan algorithm in both training and inference. The authors introduce two additional task-specific pre-post-processing operations close to Mamba that are tailored to language modelling: Hawk and Griffin. Griffin blends gated linear recurrences with local attention, aiming for both performance and efficiency. Griffin employs RG-LRU to efficiently process sequences by compressing information into a fixed-size hidden state that is iteratively updated. The gating mechanism in RG-LRU enables it to retain important information from the past while filtering out less relevant inputs, enabling the model to potentially learn long-range dependencies in the sequence. In addition to RG-LRU, Griffin incorporates local multi-query attention (Beltagy et al., 2020) to focus on a limited window of nearby tokens, while processing each part of the sequence. Hawk still uses the RG-LRU layer, but relies solely on gated linear recurrences for temporal mixing, making it a pure RNN-based model. Please refer to Sections 8.1 and 8.2 for further considerations on the Griffin model expressivity and efficiency.

Hyena Hyena (Poli et al., 2023) is a novel approach designed as a more efficient alternative to the attention mechanism prevalent in large language models. It is based on a recurrent structure, where each step involves two key components: (*i*) a long convolution operation, implicitly parameterized using feed-forward neural networks for efficiency, and (*ii*) element-wise multiplicative gating, which selectively modulates the information flow. More precisely, given three linear projections q, k, v of the input u, each of length L in $\mathbb{R}^{d_{\text{in}}}$, Hyena maps the input u_t in $(\mathcal{H}u)_t$ through,

$$(\mathcal{H}u)_t^i = u_t^i + \sum_{j=0}^{d_{\text{in}}-1} \sum_{m=0}^{t} R^{ij} q_t^j h_{t-m}^j k_m^j v_m^j, \tag{55}$$

for $i = 0, \dots, d_{\text{in}} - 1$, where h_t^j are implicit long convolution filters learned by shallow feed-forward neural networks and $R \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ is an output projection that mixes channels across the sequence length. This approach decouples filter length from parameter cost, providing advantages over explicit parameterization. The number of recurrent steps determines the complexity of the operator , and they can be represented as a decomposition of data-controlled matrices. These matrices dynamically adapt based on the input data, similar to how attention mechanisms compute a weighted sum over input elements. Instead of explicitly computing the full data-controlled matrix, Hyena leverages fast convolution algorithms, particularly Fast Fourier Transform (FFT)-based convolutions, to achieve subquadratic time complexity. Moreover, unlike some models that restrict the receptive field, it allows for interactions between any elements in the sequence through its long convolutions, enabling it to capture long-range dependencies effectively.

In subsequent work (Massaroli et al., 2023), some improvements to further enhance the efficiency of Long Convolution Sequence Models (LCSMs), including Hyena, have been introduced : the LaughingHyena distilling. It focuses specifically on improving the inference stage of these models, particularly in auto-regressive generation tasks. LaughingHyena distillation aims to represent each convolutional filter of a pre-trained LCSM as an SSM, with the smallest state dimension such that

it approximates the original filter without significant loss of accuracy. To achieve this goal, it utilizes a method called *modal interpolation*, which provides coefficients for the numerator and denominator of a rational function that minimizes the difference between the original filter and the approximating SSM transfer functions. Through these coefficients, it is possible to define the matrices A, B, and C which characterize the SSM. This distillation procedure is then followed by two steps: (i) *pre-filling* and (ii) a *recurrent update* rule. Pre-filling involves computing the state to start generating new tokens when a length-L prompt is fed to the model during auto-regressive generation, exploiting the denominator of the approximate transfer function. The recurrent update rule for the complex state is defined as follows

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t, \\ y_t &= \mathrm{Re}(Cx_t) + h_0 u_t. \end{aligned} \tag{56}$$

Here h_0 denotes the value of the original filter at initial time and $\mathrm{Re}(\cdot)$ is the real part operator (since a real-valued output is usually required). Beyond language processing, Hyena has also been employed for time series forecasting (Zhang et al., 2023) and for DNA sequence analysis (Nguyen et al., 2024).

Theoretical Foundations of SSMs Performances achieved by SSMs are remarkable, thus inspiring several research efforts to understand both their expressive capabilities and the connections to existing popular technologies (such as attention), with which they share many features but have been commonly developed in isolation. Orvieto et al. (2024) theoretically show that combining MLPs with either real or complex linear diagonal recurrences (such as in S4, Mamba, etc.) enables highly precise approximation of regular causal sequence-to-sequence maps. The proof is based on the fact that the linear RNN provides a lossless encoding of the input sequence, and the MLP conducts nonlinear processing on this encoding. While real diagonal linear recurrences are sufficient for achieving universality, employing complex eigenvalues near the unit disk, a strategy that has shown empirical success in S4, significantly improves the ability of recurrent models to store information. Cirone et al. (2024) leverage tools from Rough Path Theory, and provide theoretical grounding for the fact that, when random linear recurrences are enhanced with simple input-controlled transitions (selectivity mechanism), the hidden state is demonstrably a low-dimensional projection of a mathematical construct known as the signature of the input. This signature captures nonlinear interactions between tokens across different timescales. Other recent works focus on the connections and differences between SSMs and other sequence processing models (Hè & Kabic, 2023; Sieber et al., 2024), as long as to their links with control theory (Alonso et al., 2024). Deep SSMs such as S4, Mamba and H3 demonstrate that stacking linear recurrences with carefully chosen spectra can match or exceed attention on long contexts while retaining $\mathcal{O}(L)$ time and memory. Key themes include *(i)* diagonal, structured or low-rank transition matrices; *(ii)* convolutional or parallel-scan implementations for hardware efficiency; and *(iii)* theoretical guarantees that a linear core plus shallow non-linearity is a universal causal operator. These results confirm that recurrence–when stabilised–remains a first-class tool for sequence modelling.

The SSMs literature reveals that careful structuring of linear recurrence can recover scalability. Building on that insight, the community has revisited traditional RNNs themselves, refining gating, constraining weight geometry, or casting the dynamics in continuous time, to push their performance further. The following Section 5 reviews these architectural enhancements to "plain" recurrent networks.

5. Enhancing recurrent neural networks

This section gathers recent approaches that are not directly related to the previous macro-categories of Transformer architectures and State-Space Models, but still focus on improving recurrent models. It turns out that several of the architectural trends described in the previous sections (i.e., element-wise linear recurrence, novel gating mechanisms,

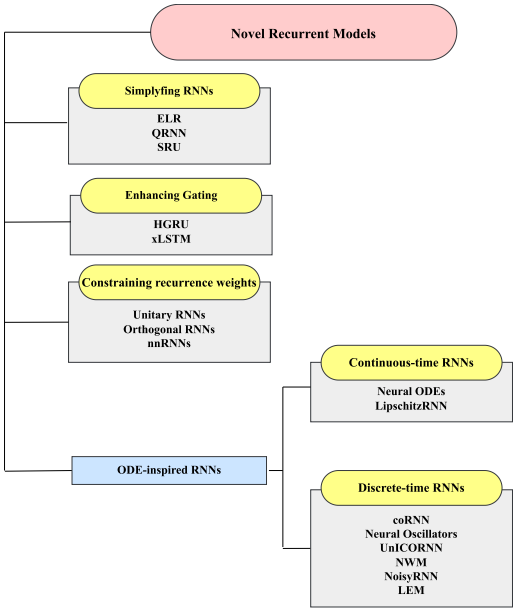

Fig. 16. Conceptual overview of the organization of Section 5, which categorizes novel recurrent models based on their architectural innovations and modeling strategies. Blue boxes indicate subsections, yellow boxes denote thematic categories, and representative models are listed in gray.

etc.), and other new ones, are also significantly explored in the scientific literature that aims to address two of the main drawbacks of RNN , namely slow sequential training and limited capability in modeling long-term dependencies. In Fig. 16, we report an overview of the main topics/approaches covered by this section.

Simplifying RNNs to Gain Speed RNNs are based on sequential processing of the input data, which does not directly allow for the development of efficient implementations that process the input tokens in parallel or update the components of the hidden state in parallel. It turns out that this limit is mostly due to (i) the non-linearity applied to recurrent layers, and (ii) the fact that updates in the hidden state are performed by full matrix multiplication, due to a dependency on all components of the hidden state from the previous time step. In detail, the standard update scheme of RNNs (Eq. (2)) assumes that all the neurons in one layer contribute to the state computation of every other neuron (i.e., through the Ax_{t-1} term). Each element of the state vector x_t depends on all the entries of x_{t-1}. Early works, such as Independently RNN (IndRNN) (Li et al., 2018), propose layers composed of "independent" neurons, achieved by modifying Eq. (2) as,

$$x_t = \sigma(a \odot x_{t-1} + Bu_t),$$

where the recurrent weight $a \in \mathbb{R}^{d_s}$ is a vector instead of a matrix, and $\odot$ is the Hadamard (element-wise) product. Notably, the gradient computed by means of BPTT, whose original form is described by Eq. (6), factorizes as $\prod_{s=j}^{t} a' \left(\sigma'(x_{s-1})\right)$, thus no matrix multiplications are involved. The authors of IndRNN derive upper/lower bound for the recurrent weight values such that IndRNN can tune the preservation or forgetting of long-term memories. Neurons in the same IndRNN layer are independent of each other, but cross-channel information over time can be propagated through multiple layers. Remarkably, assuming linear activation, a vanilla RNN with a diagonalizable recurrent weight is a special case of a two-layer IndRNN. In recent literature, recurrent layers with independent neurons and linear activation are referred to as *element-wise recurrent* (ELR) layers. Quasi-Recurrent neural network (QRNN) (Bradbury et al., 2016) deal with the inability to parallelize computation in RNNs over the temporal dimension by proposing a mixed architectures which alternates convolutional layers, working simultaneously across different time steps, and recurrent pooling functions that works in paral-

lel across different channels. QRNN alters a classical gated architectures (Hochreiter & Schmidhuber, 1997), replacing the previous hidden state x_{t-1} with the previous input u_{t-1} in the forget gate f_t computation,

$$f_t = \sigma(W_f^1 u_t + W_f^2 u_{t-1}).$$

This equation can be interpreted as a convolution with kernel-size 2 on the input sequence, an operation that can be computed in parallel along both the temporal and mini-batch dimensions. When considering larger kernel sizes, QRNN performs convolutions over the temporal dimension with a bank of filters,

$$Z = \tanh(W_z * U), \quad F = \sigma(W_f * U), \quad O = \sigma(W_o * U),$$

where $W_z, W_f, W_o \in \mathbb{R}^{d_k \times d_{in} \times d_s}$ are the convolutional filter banks with kernel size d_k, and $*$ denotes a causal masked convolution performed along the temporal dimension. Subsequently, a recurrent pooling operation computes the state, e.g., the *dynamic average pooling* with a single forget gate from Balduzzi and Ghifary (2016), $x_t = f_t \odot x_{t-1} + (1 - f_t) \odot z_t$. Simple Recurrent Units (SRUs) (Lei et al., 2018) follow the path of element-wise recurrence by substituting all the matrix-multiplications in gates with point-wise multiplications, similarly to IndRNN (Li et al., 2018). Formally, an SRU makes the cell state c_t independent and parallizable by,

$$f_t = \sigma(a_f \odot c_{t-1} + B_f u_t + b_f), \tag{57}$$

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot B u_t, \tag{58}$$

$$r_t = \sigma(a_r \odot c_t + B_r u_t + b_r), \tag{59}$$

$$x_t = r_t \odot c_t + (1 - r_t) \odot u_t, \tag{60}$$

where Eqs. (57)–(58) represent the proposed "lightweight" recurrence, where the reset gate r_t adaptively combines the input and the cell state c_t, with learnable vectors a_f, a_r, b_r, b_f and learnable matrices B_f, B_r. The skip connection in Eq. (60) favours gradient propagation. Independence between distinct hidden states enables efficient element-wise product instead of full matrix multiplication (i.e., in classical forget gates dense matrices products inject a dependencies on all neurons previous states), as the authors show when the (nonlinear) recurrence is fused within a single CUDA kernel. This allows to reduce the complexity to $\mathcal{O}(Lbd_{in})$, where b denotes here the batch dimension, while a standard LSTM takes $\mathcal{O}(Lbd_{in}{}^2)$. The seminal work by Martin and Cundy (2018) highlights that linear recurrences in the form of $x_t = \Lambda_t x_{t-1} + u_t$ are specific instances of the *scan* operation, a computation involving the repeated application of a binary operator over an array of data. This allows for a highly parallelizable unrolling of the recurrence using parallel scans (Blelloch, 1990), resulting in substantial improvements in training speeds. When Λ_t is diagonal, the cost of a ELR with parallel scan and p processors is $\mathcal{O}\big(6d_s(L/p + \log p)\big)$, while the cost of serial scan is $\mathcal{O}(2d_s L)$. This reduction becomes important when the sequence length L is large since, given sufficient processors p, the parallel time scales logarithmically with the sequence length. Please refer to Appendix H of Smith et al. (2022) for a detailed overview of the parallel scan operation. Moreover, several existing models (QRNN (Bradbury et al., 2016), SRU (Lei et al., 2018)) fall under this class of approaches, and the authors of Martin and Cundy (2018) provide an efficient CUDA kernel that speed up their training. This work laid the foundations for the efficient parallel implementation of SSMs such as S5 (Smith et al., 2022) and others (Orvieto et al., 2023) (Section 4). Additionally, while typical forget gate values depend on both the previous hidden state and the current input, the authors suggest that forget gate values should depend solely on the current inputs to enable parallel training. Lately, Lim et al. (2023) built on top of Martin and Cundy (2018) and have shown that it is also possible to parallelize the evaluation and training of *non-linear* sequential models like classic RNN and NeuralODE (Chen et al., 2018b), by introducing a general framework to solve non-linear differential equations, which are restated as fixed-point iteration problems with quadratic convergence, equivalent to Newton's method. Each fixed-point iteration involves parallelizable operations and an inverse linear operator that can be evaluated in parallel even for the aforementioned sequential models, resulting in improvements of up to 3 orders of magnitude in terms of speed in the case of long sequences.

Enhancing Gating Mechanisms Learning long-term dependencies requires the ability to modulate the effect of the incoming information. Several recent studies have incorporated gating mechanisms inspired by LSTMs (Hochreiter & Schmidhuber, 1997) (or related intuitions) into SSMs, which are characterized by gates acting on linear recurrence layers (Gu & Dao, 2023; Gu et al., 2020b; Smith et al., 2022), yielding impressive performance gains.

Gu et al. (2020b) investigated the saturation regime of gates, remarking the fact that capturing long-term dependencies in gated RNNs requires forget gate values close to one. Unfortunately, learning with gates in their saturated regimes (i.e., values close to 0 or 1) is difficult, since they suffer from vanishing gradients. Moreover, if all forget gate values are close to one, the model's ability to forget irrelevant information is hindered. To overcome such issues, the authors of Gu et al. (2020b) propose to tweak the forget gate f_t with an independent *refine gate* r_t that is exploited to produce an input-dependent bounded additive term $\phi(f_t, u_t)$ that modulates f_t, allowing much higher/lower activations,

$$r_t = \sigma(W_r^1 u_t + W_r^2 x_{t-1}),$$
$$g_t = f_t + \phi(f_t, u) := r_t\big(1 - (1 - f_t)^2\big) + (1 - r_t) f_t^2,$$
$$c_t = g_t c_{t-1} + (1 - g_t)\hat{c}_t,$$

where W_r^* denote projection matrices and $\hat{c}_t$ is the cell input activation vector from LSTMs. The form of the additive update $\phi(f_t, u)$ emerges from the consideration that gating mechanisms should be bounded in $[0, 1]$, symmetric around 0, and differentiable. Additionally, the authors propose to initialize the forget gate f_t with values sampled from the uniform distribution $\mathcal{U}(0, 1)$, instead of constant values, even allowing negative biases. This choice fosters the gate's ability to grasp different timescales, as noticed in the *chrono initialization* approach by Tallec and Ollivier (2018).

In previous Sections, we showed how several linear RNNs with static decay rates perform eigendecompositions on the recurrent weight matrix to achieve element-wise linear recurrence (Huang et al., 2022; Orvieto et al., 2023). Notice that, if only real-valued eigenvalues are allowed, this choice restricts the range of the recurrent weight matrix to be symmetric, limiting the expressiveness of the model. To overcome this limitation, linear RNNs often employ complex-valued eigenvalues, Gu et al. (2021b), Orvieto et al. (2023). Following such intuitions, Hierarchically Gated Recurrent Units (HGRU) (Qin et al., 2023c) exploit linear recurrence in the complex domain, and address the saturating issue pointed out by Gu et al. (2020b) by adding an additive learnable value Γ to the original forget gate, with the purpose of pushing gate activations away from the saturated regimes. The Γ variable, which acts as a lower bound on forget gate values, is forced to increase monotonically with the model depth, inspired from Ordered Neuron LSTM (Shen et al., 2018): small value in lower layers, in order to ease the forgetting of past-information (short-term dependencies); forget value close to one in top-most layers, facilitating the modeling of long-term dependencies. In detail, HGRU leverages a gated linear recurrent as follows,

$$\mathrm{Re}(c_t) = \mathrm{SiLU}(u_t W_{re}), \quad \mathrm{Im}(c_t) = \mathrm{SiLU}(u_t W_{im}),$$
$$f_t = \lambda_t \odot e^{i\theta},$$
$$x_t = f_t \odot x_{t-1} + (1 - \lambda_t) \odot c_t,$$

where the real (Re) and imaginary (Im) part of c_t are parametrized separately by means of learnable projections $W_\cdot$, SiLU is the Sigmoid Linear Unit function (Hendrycks & Gimpel, 2016),[18] and i is the imaginary unit. Inspired by recent works that achieve ELR by eigendecomposition of the recurrent matrix A (Gu et al., 2021b; Orvieto et al., 2023), both

[18] It is implemented as $\mathrm{SiLu}(x) = x\sigma(x)$, where σ is the sigmoid function.

the state x_t and the input mapping of HGRU are complex vectors, i.e., $x_t, c_t \in C^{1\times d_s}$, to enhance the model expressive power, as previously introduced. The magnitude λ_t of the forget gate f_t regulates the ability to retain previous information, while the phase argument θ, which is shared among time steps, determines the oscillation frequencies, in a data-independent manner. The aforementioned layer-wise increment of the additive lower bound Γ on the forget gate is achieved by acting on λ_t, as follows, assuming l is the layer index (H layers),

$$\gamma^l = [\text{cumsum}(\text{softmax}(\Gamma))]_l,$$
$$\mu_t = \sigma(B_\mu u_t),$$
$$\lambda_t^l = \gamma^l + (1-\gamma^l)\odot \mu_t,$$

where $\Gamma \in \mathbb{R}^{H\times d_s}$ independently parametrizes the lower bounds for all hidden states, where cumsum and softmax operate over the first dimension of their tensor input. Basically, the composition of the softmax and cumsum functions forms an activation function which yields a monotonically increasing vector in $[0, 1]$. The squared bracket-based notation is defined as $[\text{cumsum}(x)]_l = (\sum_{i=1}^{l} x_i) - x_1$. Notice that cumsum_0 is applied to the layer dimension across different layers, to enable upper layers to model long-range dependencies. Then, the model exploits an output projection with a learned gate, similarly to what happens in SSMs. Another perspective on the role of gating mechanisms can be appreciated by inspecting the already described connection between the update mechanisms in linear RNNs and linear Transformers, discussed in previous Sections. Recently, Zucchet et al. (2023a) proposed a unifying view on such architectures, driven by the role of gating functions. In particular, under a specific parametrization (that leverages GLU (Dauphin et al., 2017) and requires a number of parameters squared with respect to attention parameters), they showed that RNNs equipped with linear recurrent layers interconnected by feed-forward paths with multiplicative gating can, through learning, encode attention-based algorithms disguised in their weights.

Enhancing Gating Mechanisms (variants of LSTMs) The foundations of gating mechanisms, laid out by the seminal LSTM paper (Hochreiter & Schmidhuber, 1997), were recently revised, proposing two modern variants (Beck et al., 2024), referred to as sLSTM and mLSTM that, when plugged into residual backbones, yield what is referred to as xLSTM architecture. The main goal is to scale LSTMs to billions parameters to exploit them in large language models, by injecting some of the techniques we described in previous Sections, such as exponential gating and matrix-based states. For reference, we summarize in the following the cell update rules underlying vanilla LSTM,[19]

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot z_t,\\ h_t &= o_t \odot \psi(h_t),\\ z_t &= \sigma_z(W_z u_t + R_z h_{t-1} + b_z),\\ i_t &= \sigma_i(W_i u_t + R_i h_{t-1} + b_i),\\ f_t &= \sigma_f(W_f u_t + R_f h_{t-1} + b_f),\\ o_t &= \sigma_o(W_o u_t + R_o h_{t-1} + b_o), \end{aligned} \tag{61}$$

where z denote the cell input, i the input gate, f the forget gate and o output gates; W_z, W_i, W_f, W_o denote the corresponding learnable matrices connecting input u_t to the gates. R_z, R_i, R_f and R_o are the corresponding learnable recurrent weights on the hidden states and b_z, b_i, b_f, b_o are learnable biases; ψ normalizes or squashes the cell state, and is typically a $\tanh(\cdot)$, as long as the cell input activation function σ_z. Notice that the usage of recurrent weight matrices R_z, R_i, R_f, R_o allows to mix memory cells outputs. The activations σ_z, σ_i, σ_f, σ_o on gates i, f, g are typically sigmoids. The sLSTM variant introduces exponential gating on input and forget gates, in order to allow the model to better revise decisions on what to "store". Moreover, it introduces a normalizer state n_t to better stabilize the model dynamics. The first two LSTM-equations of Eq. (61) are replaced by the following three ones,

$$c_t = f_t \odot c_{t-1} + i_t \odot z_t,$$
$$n_t = f_t \odot n_{t-1} + i_t,$$
$$h_t = o_t \odot \left(c_t \odot n_t^{-1}\right).$$

The LSTM activations in Eq. (61) are implemented following specific choices: $\sigma_z := \tanh(\cdot)$ to help stabilizing the recurrence, $\sigma_i, \sigma_f := \exp(\cdot)$, $\sigma_o := \text{sigmoid}(\cdot)$. Given that the presence of exponential function could led to large values and numerical issues, the authors further stabilize gates with an additional state (please refer to the referenced paper). Additionally, sLSTMs leverage multiple memory heads, where memory mixing happens within each head (via the last introduced equations) but not across heads. As previously stated, when considering each cell in an LSTM or sLSTM, quantities (cell state, gates) are scalar (i.e., $c_t, f_t, i_t, o_t \in \mathbb{R}$), and multi-dimensionality is gained by considering h cells (i.e., $c_t \in \mathbb{R}^h$). The second model variant of Beck et al. (2024), mLSTM, enhances the vanilla model's storage scalar capacity by introducing a matrix-based cell state, $C_t \in \mathbb{R}^{d_s\times d_s}$, whose update is regulated by an outer product rule akin to Linear Transformers or Fast Weight Programmers (Schlag et al., 2021) (see Section 3). Hence, it leverages a key, query, value projections of the input as follows,

$$C_t = f_t C_{t-1} + i_t(v_t \otimes k_t), \quad n_t = f_t n_{t-1} + i_t k_t,$$
$$h_t = o_t \odot \frac{C_t q_t}{\max(n_t^T q_t, 1)}, \qquad i_t = \sigma_i(w_i' u_t + b_i),$$
$$f_t = \sigma_f(w_f' u_t + b_f), \qquad o_t = \sigma(W_o u_t + b_o),$$

where $q_t, k_t, v_t \in \mathbb{R}^{d_k}$ are linear projections of the input (such as in self-attention), while $w_i, w_f \in \mathbb{R}^{d_{\text{in}}}$ are learnable weight vectors. In the cell-state update rule, the forget gate acts as a decay rate, the input gate as a learning rate, while the output gate scales the vector which is retrieved by the outer product. The normalizer state n_t, which keeps a record of the "strength" of the gates, is the weighted sum of the key vectors, where, in turn, each key is weighted by the input gate and the current forget gate. In mLSTM, considering multiple cells is equivalent to multiple heads, since in this case there is no memory mixing, due to the presence of matrix states. Interestingly, the absence of memory mixing (no hidden-to-hidden connections, hence each cell/head can be computed independently of the others) allows us to reformulate mLSTM in a parallel form, in order to speed up training when the full sequence is available in advance (see Appendix A.3 of Beck et al. (2024) for further details). When composed into an architecture of stacked blocks, connected via residual connections of two types (with post-up projections when considering sLSTM—like Transformers—or with pre-up projections when considering mLSTM), the model is referred to as an eXtended LSTM (xLSTM). Overall, xLSTM have a $\mathcal{O}(L)$ computational and $\mathcal{O}(1)$ memory complexities. Additionally, mLSTMs, despite being computationally expensive due to the presence of matrix-based state, implements a parallel-computation form, while sLSTM is not parallelizable due to memory mixing.

Constraining the Recurrence Weights Exploding and vanishing gradients hamper the RNNs' ability to learn long-term dependencies. A recent strategy to circumvent this issue and allow the stable propagation of signals over long time scales, is to constrain the hidden-to-hidden weight matrix to be orthogonal or unitary (i.e., an element of the orthogonal group—referred to as Unitary and Orthogonal RNNs—see Arjovsky et al. (2016) and references therein), which ensures that the eigenvalues have unit norm and the dynamics are stable. However, despite advantages in terms of long-term memory preservation, this also reduces the expressivity of the model, as orthogonal transformations are limited in variety. We draw readers' attention to Salehinejad et al. (2017) (survey) and the extensive descriptions of several recent works, e.g., Erichson et al. (2021), Lezcano-Casado and Martınez-Rubio (2019). Some recent works have proposed alternative ways to overcome this trade-off, such

[19] Notice that the original model is characterized by a scalar memory cell, i.e., $c_t \in \mathbb{R}$, as processing and storage unit. Later formulations (Greff et al., 2016) combined multiple memory cells into a vector $c_t \in \mathbb{R}^h$, where h is the number of cell units. In the main text of this paper, we exploit the latter vectorial variant.

as using non-normal matrices with unit norm eigenvalues without orthogonality constraints on eigenbases (nnRNN (Kerg et al., 2019)), or formulating the recurrent units by differential equations and updating the hidden states exploiting the difference between state values (Kag et al., 2019) These methods aim to improve the performance and flexibility of RNNs while preserving their long-term memory, without explicit constraints on the weight matrices.

5.1. ODE-inspired recurrent neural networks

A recent trend involves recurrent architectures whose processing scheme is formalized by Ordinary Differential Equations (ODEs) in the context of dynamical systems. Two main branches of scientific works have developed, the former based on continuous-time RNNs and the latter on discretized ODEs.

Continuous-time RNNs Continuous-time recurrent networks have been the subject of investigation since the dawn of neural networks and, later on, they were deeply investigated at the intersection of machine learning and other scientific fields, such as signal processing (Mandic & Chambers, 2001). Amongst others, more recently, the interest in continuous-time RNNs has been renewed by studies on Neural ODEs (Chen et al., 2018b) and ODE RNNs (Rubanova et al., 2019), where a continuous ODE acts as the learning model and gradients are computed from a sensitivity equation, which allows one to trade accuracy with computing time. The state of a Neural ODEs is defined by the solutions of the equation $\dot{x} = f(x, u, t, \theta)$, where t represents continuous time, $u := u(t) \in \mathbb{R}^{d_{\text{in}}}$ the time-dependent input signal, $x := x(t) \in \mathbb{R}^{d_s}$ the RNN hidden state, $\dot{x}$ its first order time derivative and f a neural network parametrized by θ. Readers can find further details in Kidger (2022) and references therein. Liquid Time-constant Networks (Hasani et al., 2021), rather than defining the derivatives of the hidden-state directly by a neural network f as in Neural ODE, determine a more stable continuous-time RNN in the form,

$$\dot{x} = -\big(A + B \odot f(x, u, t, \theta)\big) \odot x + B \odot f(x, u, t, \theta),$$

where $A \in \mathbb{R}^{d_s}$ is a time-constant state-transition mechanism and $B \in \mathbb{R}^{d_s}$ a bias vector. Thanks to this computational structure, the neural network f determines both the derivative of the hidden state $x(t)$ and serves as an input-dependent varying time-step (i.e., dynamical, hence the term *liquid*) for the learning system.[20] Hasani et al. (2022a) computed an approximation of the solution of the integral appearing in liquid time-constant dynamics, relaxing the need for complex numerical solvers. LipschitzRNNs (Erichson et al., 2021) describe the evolution of the hidden state exploiting a functional form composed by a linear component plus a 1-Lipschitz nonlinearity (i.e., the tanh),

$$\dot{x} = \bar{A}x + \tanh(\bar{W}h + Bu + b), \qquad y = Dx,$$

where $\bar{A}, \bar{W}$ are tunable matrices with an ad-hoc fixed structure. By leveraging tools from nonlinear systems theory, the authors carried out a stability analysis on the proposed recurrent unit behaviour in the long-term, resulting in good performance and expressivity. Recently, an in-depth analysis of approximation properties and optimization dynamics of continuous-time RNNs has been carried out in Li et al. (2022c, 2020), with an interesting take on the interaction of memory and recurrent structures in the linear dynamical setting.

Discrete-time RNNs Coupled Oscillatory RNNs (coRNN) (Rusch & Mishra, 2020) leverage coupled networks of controlled non-linear forces and damped oscillators, underlying several physical systems and also in biological neurons, to ensure both expressive representations and the preservation of long-term dependencies, while constraining the dynamics of state variables and their gradients. The model is formulated through implicit-explicit time-discretizations of second-order nonlinear ordinary differential equations, capturing the dynamics of coupled oscillators in continuous time,

$$\ddot{x} = \tanh(Wx + \hat{W}\dot{x} + Vu + b) - \gamma x - \epsilon \dot{x},$$

where $t \in [0, 1]$ is the (continuous) time variable, $u := u(t) \in \mathbb{R}^{d_{\text{in}}}$ the time-dependent input signal, $x := x(t) \in \mathbb{R}^{d_s}$ the RNN hidden state RNN, and $\dot{x}, \ddot{x}$ its first and second order time derivatives; W, $\hat{W} \in \mathbb{R}^{d_s \times d_s}$, $V \in \mathbb{R}^{d_s \times d_{\text{in}}}$ are weight matrices, $b \in \mathbb{R}^{d_s}$ is the bias vector and $\gamma, \epsilon > 0$, are parameters representing oscillation frequency and the amount of damping (friction) in the system, respectively. By introducing the *velocity* variable $z := \dot{x}$ it is possible to obtain a first order system of two coupled networks defined as follows,

$$\begin{aligned} \dot{x} &= z, \\ \dot{z} &= \sigma(Wx + \hat{W}z + Vu + b) - \gamma x - \epsilon z. \end{aligned} \tag{62}$$

When discretizing such system with a fixed timestep $0 < \Delta t < 1$, the RNN hidden state at time $t_n = n\Delta t \in [0, 1]$ evolves accordingly to the following laws,

$$\begin{aligned} x_n &= x_{n-1} + \Delta t z_n, \\ z_n &= z_{n-1} + \Delta t \sigma(W x_{n-1} + \mathcal{W} z_{n-1} + V u_n + b) \\ &\quad - \Delta t \gamma x_{n-1} - \Delta t \epsilon z_{\bar{n}}, \end{aligned}$$

with $\bar{n}$ either $\bar{n} = n$ or $\bar{n} = n - 1$, depending on the fact that the damping term ϵz is treated implicitly (the former case) or explicitly (the latter). In the coupled networks defined by Eq. (62), each neuron updates its hidden state based on input signals and information from other neurons.The diagonal entries of W and the hyperparameter γ control oscillation frequency, while the diagonal entries of $\hat{W}$ and the hyperparameter ϵ determine damping for each neuron, whereas non-diagonal entries modulate interactions between neurons. Input signals drive the generation of (superpositions of) oscillatory wave-forms, controlled by the other tunable parameters. This leads to rich global dynamics and the emergence of non-trivial non-oscillatory hidden states from oscillatory inputs, emphasizing the network's high expressivity in approximating outputs from complex sequential inputs. The authors derive bounded gradients and limited hidden state magnitude for the coRNN model, under some mild assumptions. Thus, coRNN has stable dynamics which foster better performance than existing RNNs, especially on tasks with very long time-dependencies. In this family of models, which is recently referred to as Neural Oscillators (Lanthaler et al., 2023), lies UnICORNN (Rusch & Mishra, 2021), a multi-layer sequence model that stacks networks of independent (uncoupled) undamped oscillators as hidden layers within an RNN. In contrast to coRNN, neurons in UnICORNN are independent (uncoupled) and as there is no damping, the ODE system yielding UnICORNN has a Hamiltonian structure. This characteristic allows the model to avoid any assumptions on the weights, whereas the mitigation of exploding/vanishing gradients in coRNN was dependent on specific restrictions imposed on the weights. Moreover, Neural Oscillators have been proven to be capable of approximating any continuous and causal operator mapping between time-varying functions, to desired accuracy (Lanthaler et al., 2023). Their performances on long-range sequences are remarkable (Rusch & Mishra, 2021). Locally coupled oscillatory recurrent neural networks have been used to model the neuroscience concept of traveling waves, referred to as Neural Wave Machines (NWMs) (Keller & Welling, 2023). Such waves serve as a bias towards learning structured representations, which exhibit complex spatio-temporal dynamics when modeling real data. When tasked to reconstruct the input signal, NWMs use traveling waves to encode transformations in the RNN hidden state. Waves-like dynamics can be modeled also with simpler RNN architectures though connectivity constraints and initialization (Keller et al., 2023), and can act as memory storage system on complex sequence modeling. NoisyRNN (Lim et al., 2021) consider discretizations of the stochastic differential equations (SDEs) obtained from ODE formulations of RNNs through the addition of a diffusion (noise) term, as an implicit regularization. By dropping the noisy elements at inference time, NoisyRNN can be considered as a stochastic learning strategy

[20] Time-step τ_{sys} is a parameter characterizing the speed and the coupling sensitivity of an ODE. In this case, $\tau_{\text{sys}} = \frac{\tau}{1+\tau f(x,u,t,\theta)}$.

(i.e., similarly to Dropout) with several advantages such as more stable dynamics. This introduces a form of implicit regularization leading towards the development of classifiers with a large classification margin, that keep generalization error small. However, despite the stabilization properties, noise injection could negatively impact capacity for long-term memory. Rusch et al. (2021) pointed out that real-world data could contain information arranged according to multiple scales, i.e., time, lengths etc., depending on the considered data and task. They propose Long Expressive Memory (LEM), based on a time-discretization of a set of multiscale ODEs. These scales can be learned adaptively (with respect to states) and dynamically (in time). LEM has bounded gradients that mitigate exploding/vanishing issues, and favour the model ability in the context of long sequence processing. Irie et al. (2022a) introduced learning rules and Neural ODEs to build continuous-time sequence processing nets that learn to manipulate short-term memory in rapidly changing synaptic connections of other nets. This yields continuous-time counterparts of Fast Weight Programmers and Linear Transformers (Irie et al., 2021, 2022b). Learning rules can be seen as the outcome of discretization procedures applied to ODEs. Kag et al. (2019) proposed a modified differential equation for the state update, obtained by leveraging implicit discretization methods rather than the (explicit) Euler method, to foster system stability. The main intuition is that the hidden states are updated based on the difference between predicted and previous states. Then, the implicit equation is solved via fixed-point recursion, resulting in stable fixed points and fast convergence. In a subsequent work, Kag and Saligrama (2021a) also explored a time-adaptive discretization of the ODE where time steps are modified based on the current observation and the hidden state.

This section catalogued architectural innovations that aim to simplify, stabilise or extend classic RNNs: independent/diagonal updates for parallelism, unitary or orthogonal weights for gradient preservation, enhanced gating (e.g., xLSTM), and continuous-time formulations such as Neural ODEs. In the next Section 6, we summarize recently introduced efficient and forward-in-time alternatives to BPTT used to train recurrent models.

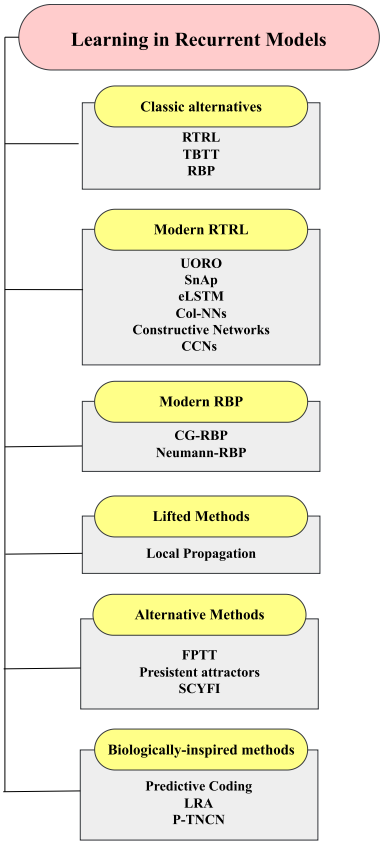

Fig. 17. Conceptual overview of the organization of Section 6, which classifies learning approaches for recurrent models based on methodological principles. Yellow boxes denote thematic categories, and representative methods are listed in gray.

6. Learning in recurrent models

Backpropagation Through Time (BPTT, see Section 2) is the de-facto standard algorithm for training recurrent models. It involves unrolling (i.e., virtually replicating) a recurrent network over the whole input sequence of length L, sharing the same parameters L times, and "back-propagating" the error from the L-th instance to the first one. In Section 2, we have emphasized the advantages and drawbacks of BPTT, such as (i) learning issues due to vanishing/exploding gradients and (ii) the high memory and computational requirements. These requirements hinder the ability to handle long-range sequences. Indeed, whenever data come in the form of a *stream* (Gunasekara et al., 2023), the BPTT requirements make it a non-feasible choice for online learning on a potentially infinite sequence, given the difficulties in unrolling the network over long time horizons. Most of the models described in previous sections propose to alleviate such issues by careful architectural designs, but still retain BPTT as the learning procedure, and assume that the whole sequence is available beforehand.

In this section, we overview recent alternative learning mechanisms that aim to address such drawbacks (i and ii) of BPTT.

We leverage a more general form of the recurrent mechanism described in Eq. (2), explicitly including a state transition function $F(\cdot)$ and an output readout function $G(\cdot)$, defined as follows. This general form simplifies the notation in the subsequent descriptions of the reviewed approaches.

$$x_t = F(x_{t-1}, u_t, \theta^F), \qquad y_t = G(x_t, u_t, \theta^G), \tag{63}$$

where, referring to Eq. (2), $\theta^F := [A, B]$ and $\theta^G := [C, D]$. Before diving into the specific details of the reviewed approaches, we showcase in Fig. 17 the organization of this section.

Classic Alternatives Eq. (5) shows how BPTT requires to cache the neuron activations for every element within the sequence processed by the model, in order to be able to perform gradient computation in the backward stage. The amount of past activations to be stored grows linearly with the sequence length, hindering BPTT usage with very long sequences. Truncated BPTT (TBPTT)(Williams & Peng, 1990) limits the gradient flow after a fixed number of time steps. While this make learning tractable in longer sequences, it inherits the structural inability to capture dependencies beyond the designated time window. In contrast, Real-time Recurrent Learning (RTRL) (Williams & Zipser, 1989) does not require the storage of past activations, and it was proposed as an *online* alternative to BPTT, which enables weight updates promptly after each new input is processed, provided that external error feedback to the model output is accessible for each input. For every timestep t, one can define the *influence* (or *sensitivity*) matrix $M_t \in \mathbb{R}^{d_s \times |\theta_F|}$ as,

$$M_t = \frac{\partial x_t}{\partial \theta^F}, \tag{64}$$

which contains the derivatives of the current state x_t with respect to the parameters θ^F of the transition function F in Eq. (63). It is possible to prove the following recurrent formula to update M_t over time,

$$\begin{aligned} M_t &= \sum_{s \le t} \frac{\partial x_t}{\partial \theta^F_s} = \sum_{s \le t-1} \frac{\partial x_t}{\partial \theta^F_s} + \frac{\partial x_t}{\partial \theta^F_t} \\ &= \sum_{s \le t-1} \frac{\partial x_t}{\partial x_{t-1}} \frac{\partial x_{t-1}}{\partial \theta^F_s} + \frac{\partial x_t}{\partial \theta^F_t} \\ &= \frac{\partial x_t}{\partial x_{t-1}} \frac{\partial x_{t-1}}{\partial \theta^F} + \frac{\partial x_t}{\partial \theta^F_t} \\ &= J_t M_{t-1} + \bar{M}_t. \end{aligned} \tag{65}$$

We now recall that the *Jacobian matrix* of a vector-valued function is a matrix of all the first-order partial derivatives of the function, w.r.t. its arguments. More precisely, let

$$f : \mathbb{R}^n \to \mathbb{R}^m, \quad f(x) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{bmatrix}. \tag{66}$$

Then, the Jacobian matrix of f with respect to x is defined as:

$$J_f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{67}$$

In Eq. (65), $J_t = \frac{\partial x_t}{\partial x_{t-1}}$ is the Jacobian of the actual state w.r.t. the previous state and $\bar{M}_t = \frac{\partial x_t}{\partial \theta_t^F}$ is called the *immediate influence*. Notice the distinction between θ_s^F and θ_t^F: the former is about instances of the weights in the previous time instants, while the latter is about the current weight values. It is possible to obtain the derivatives of the loss function with respect to θ^F by,

$$\frac{\partial \ell_t}{\partial \theta^F} = \frac{\partial \ell_t}{\partial x_t} \frac{\partial x_t}{\partial \theta^F} = \bar{c}_t M_t, \tag{68}$$

where $\bar{c}_t = \frac{\partial \ell_t}{\partial x_t}$ is called the *immediate credit assignment vector*. Differently form the just described case of θ^F, it is natural to directly learn θ^G online, because only information at present time t is required to calculate the gradient $\frac{\partial \ell_t}{\partial \theta^G}$. RTRL suggests to propagate the partials $\frac{\partial x_t}{\partial x_{t-1}}$ and $\frac{\partial x_t}{\partial \theta^F}$ from timestep t to $t+1$. This is based on the intuition that there is significant overlap in the product term (see Eqs. (5) and (6)) from time t to $t+1$, allowing for recursive computation. This algorithm is online and past-facing, which is defined by the fact that only previously computed quantities are used in the computations (Marschall et al., 2020). These properties can be directly observed in the above formulas, which explicitly depend only on timesteps t and $t-1$. RTRL is also deterministic and provides the solution in closed form. Since RTRL requires, for each time step, the storage of M_t, which involves the gradients of each component of the state (d_s elements) with respect to all the parameters involved in the state computation (i.e., $|\theta^F| = d_s \cdot d_{\text{in}} + d_s^2$), its memory complexity is $\mathcal{O}(d_s(d_s \cdot d_{\text{in}} + d_s^2)) \sim \mathcal{O}(d_s^3)$ and, for the computation of $J_t M_{t-1}$, its time complexity is $\mathcal{O}(d_s^4)$. Early attempts suffered large memory overhead limiting its usage, and while recent attempts (Marschall et al., 2020; Tallec & Ollivier, 2017) have been more successful, these methods still fall short of BPTT performance, and so trainability of RNNs is still a significant issue. Several other early attempts to solve the shortcomings of BPTT were motivated by the human way to learn from perceptual stimuli, which are intrinsically continuous over time and not pre-buffered finite-length sequences randomly shuffled to cope with stochastic gradient descent (Tiezzi et al., 2020b). Thus, from Williams and Peng (1990), the proposal of *"an on-line algorithm, designed to be used to train a network while it runs; no manual state resets or segmentations of the training stream is required"*. Even the LSTMs (Hochreiter & Schmidhuber, 1997) were introduced with a learning algorithm that unlinke full BPTT is *"local in space and time"*, where *"there is no need to store activation values observed during sequence processing in a stack with potentially unlimited size"*. Recurrent Backpropagation (RBP) (Almeida, 1990; Pearlmutter, 1995; Pineda, 1987) avoids the need to unroll the entire forward pass, as required by BPTT, by directly computing the gradient w.r.t. the learnable parameters at a steady state of Eq. (63), exploiting the implicit function theorem (Rudin et al., 1976) and achieving constant memory complexity w.r.t. the number of processing steps. In details, RBP assumes that the dynamics of the state transition $F(\cdot)$ reach an equilibrium with a steady-state hidden state x^*, i.e., $x^* = F(x^*, u, \theta^F)$ when processing an input u, which is fixed and not time-dependent.[21] In this condition, it is possible to construct a function $\Psi(x, \theta^F) = x - F(x, u, \theta^F)$ such that, when the system dynamic has reached the equilibrium (i.e., a fixed point of the state, x^*), $\Psi(x, \theta^F) = 0$. Differentiating $\Psi(x, \theta^F)$ w.r.t the parameters θ^F at x^* and rearranging the terms yields the gradient of the steady state x^* w.r.t. the parameters of the stable dynamical system,

$$\frac{\partial x^*}{\partial \theta^F} = \left(I - J_{F,x^*}\right)^{-1} \frac{\partial F(x^*, u, \theta^F)}{\partial \theta^F}. \tag{69}$$

where $J_{F,x^*} = \frac{\partial F(x^*, u, \theta^F)}{\partial x^*}$ is the Jacobian matrix of F evaluated at x^*. This is a result from the Implicit Function Theorem (Rudin et al., 1976), which requires (i) Ψ to be continuously differentiable and (ii) $I - J_{F,x^*}$ to be invertible (Liao et al., 2018). The term $\frac{\partial x^*}{\partial \theta^F}$ is exploited to compute the gradient of the loss function w.r.t. the learnable parameters that, by leveraging the chain rule, is,

$$\frac{\partial L}{\partial \theta^F} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x^*} \frac{\partial x^*}{\partial \theta^F}. \tag{70}$$

When substituting Eq. (69) into Eq. (70) loss derivative, we get,

$$\frac{\partial L}{\partial \theta^F} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x^*} \left(I - J_{F,x^*}\right)^{-1} \frac{\partial F(x^*, u, \theta^F)}{\partial \theta^F}. \tag{71}$$

Given that the Jacobian is non-symmetric for standard RNNs, directly using solvers for linear system is impractical. Conversely, the standard RBP approach is to compute the term,

$$z = \left(I - J_{F,x^*}^T\right)^{-1} \left(\frac{\partial L}{\partial y} \frac{\partial y}{\partial x^*}\right)', \tag{72}$$

via fixed point iterations. RBP was the learning algorithm exploited in the seminal works introducing Graph Neural Networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008), which can be considered generalizations of RNNs to handle graph-structured input data (Frasconi et al., 1998; Micheli, 2009; Sperduti & Starita, 1997). Indeed, the current literature refers to GNNs by Scarselli et al. (2008) as Recurrent GNNs (RecGNNs) (Wu et al., 2020). Inference in RecGNNs can be interpreted as a diffusion process along the graph up to the convergence to fixed points of the nodal states. RecGNNs ensure the RBP conditions (i) and (ii) by forcing $F(\cdot)$ to be a contraction map on a Banach space, a choice thatneverthelessposes some strong limitations on the model capacity , and is difficult to satisfy. When used with models that satisfy such assumptions, the main computational cost of RBP lies in solving a linear system (i.e., where the most expensive operation is the matrix-vector product $J_{F,x^*}^T z$), which has constant memory and computation time with respect to the number of unrolling steps.

Modern RTRL Because of the high memory and time complexity characterizing RTRL, researchers have recently focused on finding more efficient approximations. For example, Unbiased Online Recurrent Optimization (UORO) (Tallec & Ollivier, 2017) is a stochastic approximation of RTRL. If, for simplicity in the description and without any loss of generality, we consider θ^F to be a matrix of weights, indexed by pairs of indices (i, j), UORO decomposes M_t of Eq. (64) (which is a 3D tensor due to what we just stated about θ^F) as the outer product of two tensors of lower rank A_t and B_t, such that,

$$M_t^{kij} = A_t^k B_t^{ij}, \tag{73}$$

being k the index of a unit belonging to the state. UORO provides stochastic approximations for A_t and B_t defined through a random vector $\nu \in \mathbb{R}^{d_s}$, which satisfies $\mathbb{E}[\nu^i \nu^j] \propto \delta_{ij}$ and $\mathbb{E}[\nu^i] = 0$. More precisely,

$$\begin{aligned} A_t^k &= \rho_0 \sum_{k'} J_t^{kk'} A_{t-1}^{k'} + \rho_1 \nu^k, \\ B_t^{ij} &= \rho_0^{-1} B_{t-1}^{ij} + \rho_1 \sum_{k'} \nu^{k'} \bar{M}_t^{k'ij}, \end{aligned} \tag{74}$$

[21] RBP formulation assumes a fixed not time-dependent input u. However, in common scenarios where data is i.i.d, the sequential input data can be interpreted as sampled from a stationary distribution. As a consequences, RBP can be applied, since the steady state holds in expectation (Liao et al., 2018).

where ρ_0 and ρ_1 are positive constants. It is possible to prove (see Marschall et al. (2020)) that the resulting outer product is an unbiased estimator for M_t. This algorithm is still online and past-facing, but the memory and time complexities are reduced to $\mathcal{O}({d_s}^2)$. Another example of RTRL approximation is the Sparse n-Step Approximation (SnAp) (Menick et al., 2020) algorithm, which imposes sparsity on the matrix M_t to reduce the amount of computation in the product $J_t M_{t-1}$ of Eq. (65). The "n-Step" in SnAp refers to the fact that the algorithm considers gradients over n time steps for approximating M_t. More precisely, if θ^F is flattened as a vector, the influence matrix M is approximated as follows,

$$M_t^{kz} \approx \begin{cases} M_t^{kz}, & \text{if } \theta_t^{Fz} \text{ influences hidden unit } x_{t+n}^k \\ 0, & \text{otherwise.} \end{cases} \tag{75}$$

If we indicate with s the level of sparsity of the matrix M_t, and we define d as $d := 1 - s$, the SnAp-1 algorithm ($n = 1$) for a fully connected RNN has a memory complexity of $\mathcal{O}(d_s + d|\theta^F|)$ and a time complexity of $O(d({d_s}^2 + |\theta^F|))$. RTRL can be also made computationally tractable introducing different RNN structures and adopting specific learning processes exploiting these architectures for enabling scalable, unbiased and noise-free gradient estimation. For example, Irie et al. (2023b) proposed to apply RTRL gradient computation to RNNs with element-wise recurrence (eLSTM). Under this architectural assumption, it is possible to obtain forward recursive formulas for the inference matrix, which have a memory complexity of $\mathcal{O}(d_{\text{in}} d_s)$ and a per-step time complexity of $\mathcal{O}({d_s}^2)$. Always focusing on the net structure, Sutton et al. proposed Columnar Neural networks (Col-NNs) (Javed et al., 2021b), Constructive Networks and Columnar-Constructive networks (CCNs) (Javed et al., 2021a, 2023). In Col-NNs (Javed et al., 2021b), The network structure is restricted to be composed of independent, potentially deep columns. Each column presents a scalar recurrent state, which is not connected to the states of the other columns. Therefore, it is possible to apply RTRL to each column individually, reducing the computational cost to $\mathcal{O}(|\theta_F|)$, which means that RTRL for Col-NNs scales linearly with the size of the parameters. However, the structure of Col-NNs lacks hierarchical recurrent features. To introduce this hierarchy, Constructive Networks (Javed et al., 2021a) have been introduced, learning the recurrent network one feature at a time. The learning process prioritizes the acquisition of weights associated with the first recurrent feature, exclusively linked to the input. Subsequently, these weights are frozen, facilitating progression to the subsequent hidden unit. This unit can now establish connections with preceding recurrent features. Notably, the output weights remain dynamic, undergoing continual updates. By sequentially focusing on discrete subsets of the network during training, Constructive Networks incur even lower per-step computations than Columnar Networks. Consequently, RTRL can be implemented efficiently. Constructive Networks have the limitation of being unable to learn multiple features in parallel. To overcome this issue, CCNs (Javed et al., 2023) learn multiple columns that take as input the features of all the existing frozen columns, iteratively constructing multiple columns of recurrent features in parallel. For other approximations of RTRL, see Marschall et al. (2020). Finally, inspired by the success of networks with recurrent units (Orvieto et al., 2023) and leveraging the fact that recurrence with only self-loops greatly simplifies RTRL (Mori et al., 1989; Mozer, 2013), in Zucchet et al. (2023b) the authors propose a modification of RTRL that is tailored to architectures with linear recurrent blocks interconnected through nonlinear layers. In particular, within this setting, they propose an approximation of the gradient in the case where more than one recurrent layer is stacked, artificially reducing the dependencies over time across layers.

Modern RBP Liao et al. (2018) investigated the strict requirements of RBP and proposed two variants based on conjugate gradient on the normal equations (CG-RBP) and Neumann series (Neumann-RBP), respectively. The former exploits an iterative solver to tackle Eq. (72), the conjugate gradient method on the normal equations, that however requires an expensive matrix multiplication (i.e., $J_{F,x^*} J_{F,x}^T z$) and, given that it uses normal equations, has a squared condition number leading to slower convergence times. The latter, Neumann-RBP, exploits a property of convergent Neumann series, $\sum_{k=0}^{\infty} A^k = (I - A)^{-1}$. A sufficient condition for the series convergence is that the largest absolute eigenvalue of A, namely λ, must be $\lambda < 1$. When $A = J_{F,x}^T$, it is possible to replace the $(I - J_{F,x^*}^T)^{-1}$ term in Eq. (71) with the series sum. Additionally, the gradient $\frac{\partial L}{\partial \theta^F}$ can be approximated with the K-th order truncation of the Neumann series. Memory complexity is constant with respect to the number of truncation steps, and the algorithm, given that it relies only on the steady state x^*, does not require to store the hidden states in the forward pass of the RNN, as done in BPTT. The authors remark the equivalence of Neumann-RBP with BPTT when the Neumann series converges, and that a K-step Neumann-RBP is equivalent to a K-step Truncated BPTT. Linsley et al. (2020) focused on recurrent vision models (Linsley et al., 2018) and remarked that when trained with standard RBP their training dynamics devolve into an unstable regime. The authors identified the root cause of this issue in the aforementioned condition (*ii*) of RBP, i.e., the fact that $I - J_{F,x^*}$ is not invertible. Forcing $F(\cdot)$ to be a contraction map, as in Scarselli et al. (2008), requires globally contractive model components, such as squashing nonlinearities (e.g., sigmoid and tanh), that however can be suboptimal for some computer vision tasks and hinder recurrent vision model performances. Thus, the authors proposed a soft architecture-agnostic constraint for learning local contraction maps, i.e., the Lipschitz Coefficient Penalty (LCP) $\|(\mathbb{1} \cdot J_{F,x^*} - \lambda)^+\|_2$, where $(\cdot)^+$ denotes element-wise rectification and $\lambda \in [0, 1)$ is an hand-selected Lipschitz constant which bounds $\|J_{F,x^*}\|_2$, tuning the degree of contraction in $F(\cdot)$. This choice allows to keep the largest singular value of $J_{F,x^*} < 1$, and forces $F(\cdot)$ to be locally contractive at x^*. LCP can be combined with any task-related loss function for optimization.

Lifted Methods The vanishing/exploding gradient issues that arise in training RNNs with BPTT are critical due to the usage of gradient descent as the optimization technique. Alternative approaches have emerged within the family of *lifted methods*. Overall, the main intuition of lifted methods is to act in an enlarged space where the neural states represent additional variables to the training problem, and the propagation dynamics of Eq. (2) are expressed as architectural constraints as follows,

$$\begin{aligned} &\min_{\theta,x} \quad \mathcal{L}(\theta) \\ &\text{s.t.} \quad\ \ x_t = F(x_{t-1}, u_t, \theta^F), \end{aligned} \tag{76}$$

where $\mathcal{L}$ is the loss function of Eq. (4). Thus, such models are referred to as "lifted" because the usual parameter search space of θ-variables is lifted to an higher dimensional space composed by (θ, x)-variables. The common approach is to transform the non-smooth constrained optimization problem into a smooth unconstrained problem in the enlarged space. Early approaches were proposed for feed-forward architectures (but can be easily extended to RNNs) (Carreira-Perpinan & Wang, 2014), by adding quadratic penalties to approximately enforce the equality constraints. Succeeding works (Taylor et al., 2016) use Lagrange multipliers to exactly enforce equality constraints and optimize via the Alternating Direction Method of Multipliers (ADMM) and Bregman iteration. A clear advantage of these methods is the ability to decompose the training problem into multiple, local sub-problems, which can be solved efficiently. Recently, works by Askari et al. (2018), Gu et al. (2020c) proposed convex and biconvex formulations that can be optimized using Block Coordinate Descent. These methods were extended to RNNs in Askari et al. (2018). Marra et al. (2020) investigated the connections of lifted methods to Backpropagation, and proposed a hard-constraining scheme, referred to as Local Propagation, based on the augmented Lagrangian. Learning consists of a differential optimization problem converging towards a saddle point of the Lagrangian. Interestingly, this approach has been extended to devise a novel learning algorithm for RecGNNs (Tiezzi et al., 2020a), allowing for the exploitation of deep RecGNNs (Maggini et al., 2024; Tiezzi et al., 2021), by implicitly ex-

pressing the state convergence procedure via a constraint satisfaction mechanism. This removes the need for iterative procedures to be applied at each training epoch, and eliminates the network unfolding of the original model and the harsh constraints imposed by RBP. Despite achieving good performance in classic benchmarks, the memory requirements of lifted methods, due to the introduction of additional trainable parameters, remain their main drawback.

Alternative Methods There have been several other attempts to propose alternatives to BPTT. Forward Propagation Though Time (FPTT (Kag & Saligrama, 2021b)) avoids the temporal unrolling by updating the RNN parameters at each time step towards the optimization of an instantaneous risk function: the loss at time t plus a dynamically evolving regularizer, controlled by a state-vector, which summarizes the evaluation of past losses. In details, at each time t, a two-step update is applied, given the instantaneous loss function $\ell_t(\theta) := \ell\left(y_t, \hat{y}_t|\theta\right)$ and the RNN learnable parameters θ (see Section 2):

$$\begin{aligned}\overline{\ell}(\theta) &:= \ell_t(\theta) + \frac{\alpha}{2}\|\theta - \overline{\theta}_t - \frac{1}{2\alpha}\nabla\overline{\ell}_{t-1}(\theta_t)\|^2,\\ \theta_{t+1} &= \theta_t - \eta\nabla_\theta\overline{\ell}(\theta)\Big|_{\theta=\theta_t},\\ \overline{\theta}_{t+1} &= \frac{1}{2}(\overline{\theta}_t + \theta_{t+1}) - \frac{1}{2\alpha}\nabla\ell_t(\theta_{t+1}),\end{aligned} \tag{77}$$

where α is a weighing factor, $\overline{\ell}(\theta)$ denotes the augmented instantaneous risk function and $\overline{\theta}$ is the state-vector that summarizes past losses. Indeed, such a "summary" represents a running average of past θ_t plus a correction term, that enforces stability in updates and convergence of the parameters toward a fixed point (i.e., only in this case hidden state trajectories simulate the usual static time invariant RNN parameters). FPTT requires $\mathcal{O}(L)$ gradient computations for an L-length sequence, while BPTT computes gradient only once, when the whole sequence has been processed. However, the constants (when evaluating the complexity) involved in taking gradient for the full-sequence are higher than computing single-step gradients. From the memory point of view, BPTT stores intermediate hidden state about the whole sequence, i.e., $\mathcal{O}(L)$, while FPTT does not require storing hidden states, i.e., it is $\mathcal{O}(1)$. The authors of FPTT additionally propose a more efficient variant that performs updates restricted to K time-steps, referred to as FPTT-K. Given that Eq. (77) exploits instantaneous loss computations, when the task does not involve step-wise supervisions (e.g., terminal prediction), FPTT leverages an alternative formulation that approximate the previous one (Kag & Saligrama, 2021b).

Going beyond FPTT, another alternative method for addressing the exploding-vanishing gradient problem in processing long-sequences, is the one of Park et al. (2023). Such a method consists in a novel initialization scheme for RNNs enhancing learning of temporal dynamics. In order to achieve a *stable limit cycle*, defined as an attracting ring manifold where the neural activations form a periodic trajectory, the authors structure the weight matrix using a scaled rotation matrix. This results in a block orthogonal matrix arrangement where, within each 2×2 block, the behavior of the paired neurons exhibits spontaneous oscillations. The exploding-vanishing gradient issue can also be related to the presence of *bifurcations* in the parameter space of the RNN, which represent qualitative shifts in network dynamics due to parameter variations. As shown by Eisenmann et al. (2023), bifurcations are associated with abrupt changes in stability regions and the topological structure of the state space. The authors of Eisenmann et al. (2023) propose a heuristic algorithm (SCYFI, which stands for Searcher for Cycles and Fixed points) for identifying fixed points and cycles in ReLU-based RNNs and determining their existence and stability regions in the parameter space, along with eventual bifurcations. Once that fixed points, cycles and bifurcations have been identified, Generalized Teacher Forcing (GTF) (a method aimed at redirecting diverging trajectories towards their intended targets) tends to circumvent bifurcations during training. Finally, we mention the different route followed by Echo State Networks (ESNs) and, more generally, by instances of Reservoir Computing (RC), where a large (usually non-learned) sparse connectivity layer is followed by a trainable readout function. ESN offer a compelling alternative to traditional RNNs by separating the roles of dynamic feature extraction and learning. In an ESN, the core of the network, called the *reservoir*, consists of a large, fixed, and sparsely connected recurrent layer, whose weights are typically randomly initialized and left untrained. This reservoir serves as a dynamic projection space that transforms input signals into a rich set of nonlinear features over time. The key innovation lies in *echo state property*, which ensures that the effect of any initial state fades away, making the network response dependent primarily on the recent input history. This property allows for stable dynamics without the need to train the internal connections, significantly simplifying the learning process compared to standard RNNs. Learning in ESNs is confined to the readout layer, a typically linear function that maps the high-dimensional reservoir states to the desired output. Because only this readout is trained, usually via efficient linear regression techniques, ESNs are computationally efficient and less prone to issues like vanishing or exploding gradients. The reader can find more information in Gallicchio and Micheli (2011), Micheli and Tortorella (2022), Scardapane et al. (2023).

Koopman operator view Koopman theory provides a powerful mathematical framework for analyzing nonlinear dynamical systems by embedding them into a linear, though typically infinite-dimensional, function space. The core idea is to focus not on the system's state evolution itself but on how observable functions of the state evolve over time under the action of the *Koopman operator*, which is linear even if the underlying system is nonlinear (Koopman, 1931). This approach has found recent traction in machine learning, particularly for modeling time-series data, where traditional RNNs struggle with issues like vanishing gradients and lack of interpretability (Bengio et al., 1994; Pascanu et al., 2013). In this context, Azencot et al. (2020) propose a *Consistent Koopman Autoencoder*, a model that draws a direct link between Koopman theory and deep learning by embedding high-dimensional temporal data into a latent space where dynamics evolve linearly via learned Koopman operators. Unlike standard RNNs, which rely heavily on training to learn latent state transitions, this method uses physics-informed constraints, such as forward-backward consistency, to promote stability and interpretability. Their approach retains computational tractability while outperforming traditional and Koopman-based baselines, especially under noise and over longer horizons. Thus, Koopman theory not only complements RNNs by offering a principled alternative but also enriches neural forecasting architectures with physically meaningful structure and improved generalization capabilities.

Biologically-inspired Methods Despite its extensive and successful exploitation in training procedures for state-of-the-art models, the Backpropagation algorithm (BP) has been also criticized about its lack of biological plausibility (Crick, 1989). The main issue is represented by the way in which BP performs credit assignment (see Section 2), with backpropagation-based synaptic weights adaptations that are unlikely to occur in real neuronal cells. Several other issues such as weight transport, non-locality, global feedback pathway, are summarized in recent works (Lillicrap & Santoro, 2019; Millidge et al., 2022; Ororbia, 2023). BPTT inherits all the plausibility-related issues from BP, with the addition that the need to store neural activities for error computations is further exacerbated by the requirement to store them over time—a very different procedure w.r.t. the local message-passing structure happening in the brain (Ororbia, 2023). Among several biologically plausible synaptic modification rules, such as Hebbian Learning (Kuriscak et al., 2015), Millidge et al. (2022) recently proved that Predictive Coding (Whittington & Bogacz, 2017) approximates BP in arbitrary computational graphs, including RNNs, by relying solely local and plausible rules. Predictive Coding revolves around the perspective that the brain operates as a probabilistic generative model, consistently formulating predictions about its surroundings and adjusting internal hypotheses in accordance with sensory data, in line with the Bayesian brain hypothesis (Ororbia, 2023). In Millidge et al. (2022), the variational inference problem to be solved is that of inferring activation values of each node

Table 5
Comparative summary of two key features of training algorithms for recurrent models. We indicate if an algorithm is able to learn online on single components of each sequence, if the overall complexity (either spatial or temporal or both) is $\mathcal{O}(L)$ (assuming all other parameters fixed), and if it scales worse than $\mathcal{O}(d_s^2)$ in term of its dependence from the hidden state size. See the paper text for more details.

Algorithm	Online	$\mathcal{O}(L)$?	$\mathcal{O}(d_s^k)$, $k > 2$?
BPTT (Rumelhart et al., 1985)	No	Yes	No
RBP (Almeida, 1990)	No	Yes	No
RTRL (Williams & Zipser, 1989)	Yes	No	Yes
UORO (Tallec & Ollivier, 2017)	Yes	No	No
SnAp-1 (Menick et al., 2020)	Yes	No	No
FPTT (Kag & Saligrama, 2021b)	Yes	Yes	No
Local Prop. (Marra et al., 2020)	Yes	No	No

in the computational graph (i.e., activation values of neural units) given the start nodes (the input data) and target nodes. The neural computational graph is enriched by *error neurons/units* that encode the difference between the activity at a given layer of the network and that which is predicted or generated from its parent nodes. In the case of RNNs, both the output y_t and the states x_t are augmented with additional error units that compute the mismatch between the actual values of the corresponding variables and the prediction obtained via variational inference. The final goal is the optimization of a functional known as the variational free energy, aiming at improving the generative model accuracy and minimizing model complexity.

Inspired from the same notions, Local Representation Alignment (LRA) (Ororbia & Mali, 2019) aims at optimizing a related variational free energy by minimizing the discrepancy between neural activations and ad-hoc local targets. LRA does not assume an underlying probabilistic generative model, and associates each layer with a target activity vector, so that synaptic weights are modified to guide the layer activity towards these targets. LRA ensures that realistic targets are chosen, considering the possible representation of a layer. Mismatch signals are computed as the first derivative of a distance function, and the target representation for each layer is determined based on accumulated mismatch signals from other layers. The synaptic plasticity rules involve adjustments to connections between layers and error message passing pathways. The effectiveness of training recurrent models with LRA has been explored in Ororbia et al. (2020), where the authors propose the Parallel Temporal Neural Coding Network (P-TNCN), a layerwise parallelizable recurrent model which avoids unfoldings over the temporal dimension. The reader can refer to the very recent survey (Ororbia, 2023) for a comprehensive overview on biologically plausible learning rules.

In summary, beyond architecture, recent work revisits training itself: modern approximations to RTRL, local credit-assignment schemes, lifted optimisation, and biologically inspired predictive-coding variants all aim to circumvent BPTT's memory footprint and latency. These algorithmic developments complement the architectural trends surveyed earlier, completing the picture of next-generation recurrent sequence processing. Table 5 reports a comparative summary of the algorithms mentioned in this section, highlighting their key properties and complexities. In the next Section 7, we review the most popular benchmarks for evaluating the models surveyed so far.

7. Benchmarks

Devising appropriate benchmarks for evaluating the wide spectrum of architectures for processing long sequences is a fundamental step. Indeed, looking back at the whole history of sequence processing, up to the present day, there is a clear lack of a well-established and consistent consensus on which benchmarks to use when comparing different algorithms. Ideally, benchmarks should create conditions to investigate a model's ability to preserve long-term dependencies, providing long sequences and tasks in which old information might play an important role. However, the meaning of *long sequence* has changed over the years. While in the early era of vanilla RNNs and LSTMs, a sequence was considered to be "long" when it consisted a number of steps in the order of a few tens or hundreds, current models (Recurrent Transformers, SSMs) are capable of processing sequences composed of thousands to millions of tokens (Bulatov et al., 2023).

In this section , we describe the most common benchmarks that have been tackled since the dawn of RNNs up to the more recent Long Range Arena benchmark (Tay et al., 2020), and our analysis is aimed at providing a comprehensive view of benchmarking with long sequences. We mostly focus on the benchmarks and datasets used in the large set of papers described in this survey: Table 8 summarizes the benchmarks (first column) and the scientific papers in which such benchmarks have been used (subsequent columns, each column containing the papers described in a different section of this survey). We categorized the various benchmarks according to their application field, distinguishing among synthetic benchmarks, computer vision, natural language, time series, audio/speech, and the recent Long Range Arena (Tay et al., 2020), which includes multiple sub-benchmarks.

7.1. Synthetic benchmarks

Adding Task The Adding task (Hochreiter & Schmidhuber, 1997) is a sequence-based regression problem which serves as a benchmark to evaluate the capability of RNNs in learning long-term dependencies. At every time step, a 2-dimensional input vector is provided, $u(t) = (u_1(t), u_2(t))$, consisting of a real-valued signal, sampled uniformly in range $(0, 1)$, and a binary value, respectively. In particular, given a sequence of L elements, $u_2(t) \neq 0$ only in two specific time instants of the L ones that compose input sequence. Let us assume such two time instants are t_a and t_b. The goal is to predict the sum of the real-valued entries of the inputs proved at t_a and t_b, i.e., $u_1(t_a) + u_1(t_b)$ (see Fig. 18). The challenge of this task consists in capturing the possibly long-term dependencies between the pairs of relevant elements of the sequences. Sequences have different lengths, usually $L \in (100, 500, 1000)$. The quality of a model in approaching this task is evaluated in function of its ability to accurately predict the aforementioned sum, measured by the Mean Squared Error (MSE). The baseline performance is set by a simple model always predicting a sum value of 1, resulting in an MSE of ≈ 0.1767. This task has been exploited in several works (Arjovsky et al., 2016; Hochreiter & Schmidhuber, 1997; Kag & Saligrama, 2021b; Le et al., 2015; Li et al., 2018) to assess the capability of the proposed RNNs in handling long-range dependencies.

Copy Task The Copy task (Hochreiter & Schmidhuber, 1997) is a largely known benchmarks to evaluate the network ability to recall information observed several time steps in the past, asking the network to reproduce the "initial" tokens of a long sequence. Data consist of sequences of $L + 20$ digits. The first ten tokens $(a_0, a_1, \dots, a_9)$ are randomly chosen from $\{1, \dots, 8\}$ and represent the sequence to be "remembered". The subsequent L tokens are populated by noise (usually simply modeled by the integer 0), and the last ten tokens are all equal to the *repeat marker* (usually the integer 9), that serves as an trigger to tell the model that it is time to reproduce the initial 10 tokens. In fact, the goal of the recurrent model is to output the starting ten tokens $(a_0, \dots, a_9)$ in the correct order during the last 10 time steps, whenever the *repeat marker* 9 is presented. The random guessing baseline has loss $\log(8) \simeq 2.08$. Training and test examples are both generated this way. Intuitively, temporal credit assignment is critical in this task. Several relevant works tackled this task (Arjovsky et al., 2016; Ba et al., 2016b; Gu et al., 2020a; Hochreiter & Schmidhuber, 1997; Menick et al., 2020; Park et al., 2023; Zucchet et al., 2023b). Gu and Dao (2023) observed that Linear Time Invariant SSMs (Section 4), encompassing linear recurrence and global convolutions, can efficiently tackle this task by focusing

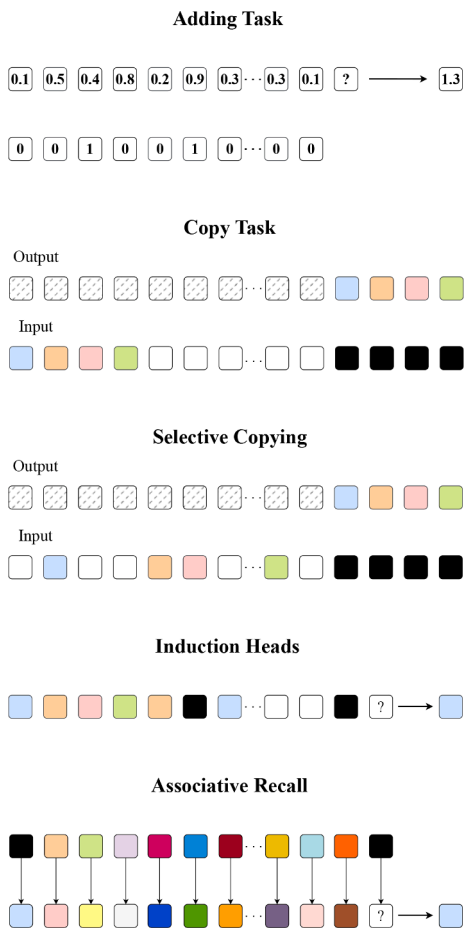

Fig. 18. Synthetic benchmarks. The standard version of the Copy task involves constant spacing between input and output elements and is easily solved by time-invariant models exploiting linear recurrences and global convolutions. Crossed-out tokens in the output denote tokens that are masked out in the loss. The Selective Copying task has random spacing in between inputs and requires time-varying models that can selectively remember or ignore inputs depending on their content. The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for Large Language Models. See the main text for further details.

solely on tracking time rather than comprehensively understanding the data. For instance, this could involve constructing a convolution kernel of the right length, as also pointed out by Romero et al. (2021). For this reason, it was proposed to exploit the Selective Copying task (originally introduced by Jing et al. (2019) as the Denoise task). This task disrupts the straightforward approach of Copy by introducing random spacing between relevant tokens (i.e., relevant tokens are interleaved by a varying number of noise-like tokens, see Fig. 18), eliminating trivial solutions. Consequently, this adjustment necessitates content-aware reasoning for the network to effectively memorize and filter out relevant tokens while disregarding irrelevant ones. Lately, the Selective Copy task has been used to asses Large Language models capability of retraining long range dependencies (De et al., 2024; Gu & Dao, 2023), with sequences of $L = 4096$, and a vocabulary of 16 tokens. Some other variants, such as the Capacity task, have been proposed to evaluated the case of sliding windows (Voelker et al., 2019).

Induction Heads In-context learning abilities of Large Language Models (LLMs)[22] have been recently hypothesized to emerge from the so-called induction head mechanism, that was tested by means of the synthetic Induction Heads task, proposed in Olsson et al. (2022) (Fig. 18). In this task, models are required to perform copy and *associative recall* operations. Specifically, the models are tasked with recalling the token immediately following a designated special token (referring to Fig. 18: the black token is the special token, the token to be retrieved is the blue one). When the special token is presented once again, the model must be capable to subsequently retrieve and output the token following it within the context (the blue token in Fig. 18). The successful execution of this task implies that the model possesses the ability to generalize to longer sequences than those it was trained on. During training, a vocabulary size of t tokens is exploited ($t = 16$ in De et al. (2024), Gu and Dao (2023), various t are tested in Poli et al. (2023)), with sequences of length $L = 256$, composed by tokens randomly sampled from the vocabulary. Additionally, the location of the special token within the sequence is also randomly determined. The model generalization and extrapolation abilities are assessed at test time on sequences with increasing lengths, ranging from $2^6 = 64$ to $2^{20} = 1,048,576$. Induction Heads is exploited in several recent works (De et al., 2024; Gu & Dao, 2023).

Associative Recall Proposed in Ba et al. (2016a), this task is similar to Induction heads but introduces an additional challenge. The model processes multiple key-value pairs in a sequence (keys are characters, values are digits—the sequence is composed by keys followed by values, see Fig. 18). At the end of the sequence, one of the keys is presented as query, and the model must recall the specific value associated to it. It has been used in Bulatov et al. (2022), Dao et al. (2023), Gu and Dao (2023), Poli et al. (2023), Yang et al. (2023). Arora et al. (2023) proposed an harder Multi-query Associative Recall task where multiple queries must be recalled.

Phonebook Look-Up In the recently introduced PhoneBook task (Jelassi et al., 2024), the model is asked to process a synthetic phone-book (e.g., each line looks like "John Powell: 609-323 7777") and to return the phone number when given a name. Various sizes of the phone-books are tested. The main intuition behind this task is to prove that despite the efficiency at inference-time, SSMs are somewhat limited when compared to Transformers on tasks that require copying from the input context, as assessed by De et al. (2024), Jelassi et al. (2024).

Other Synthetic Tasks The seminal paper proposing LSTMs by Hochreiter and Schmidhuber (1997) contains several interesting tasks to test long-range dependencies. Among the others, in Task 2b the input consists of sequences of length L, composed of vectors $u \in R^{d_{\text{in}}}$. The input at $t = 0$ is either $u_0 = (1, 0, \dots, 0)$ or $u_0 = (-1, 0, \dots, 0)$. For $t > 0$, u_t is a randomly chosen one-hot vector. The evaluated model must process the entire sequence and then output the sign of the first component of u_0. Recent works exploited these benchmarks (Li et al., 2018). Bulatov et al. (2022) proposed some other variants. For instance, in the Reverse Task, a certain input sequence should be generated at test time in the reversed order.

7.2. Computer vision benchmarks

MNIST The MNIST dataset (LeCun et al., 1998) has been widely exploited to asses performances of sequence processing models. There are two common alternative setups/tasks: sequence classification and generative autoregressive image modeling. In the former, each MNIST digit (28×28 pixels) is decomposed into a sequential pattern and classification is performed once the whole sequence has been processed. In the scientific literature, different modalities have been explored[23], i.e., processing each MNIST digit following the order of scanlines, one pixel at

[22] In-context learning is usually defined as the ability to learn directly from examples within the context (the prompt) of the input sequence at inference time, and use information from the input to generate the right answer for the output (Olsson et al., 2022).

[23] This also depends on the model architecture that is being tested. An RNN processes vectorial patterns (pixel-by-pixel or row-by-row), while Transformer-based models generally follow the Visual Transformer (ViT) (Dosovitskiy et al., 2020) approach of processing a sequence of image patches.

a time, starting at the top-left corner of the image, and ending at the bottom-right corner (resulting in a sequence with $L = 784$)—or alternatively, row-by-row, where each row represents the model input at a certain time t, resulting in $L = 28$). We refer to both these dataset variations as Sequential MNIST (sMNIST) (Le et al., 2015). An harder variant is permuted sequential MNIST (psMNIST), where the same fixed random pixel permutation is applied to each digit of the dataset, resulting in an increased time-delay between interdependent pixels. The datasets sMNIST and psMNIST are exploited in Gu et al. (2021b), Kag and Saligrama (2021b), Le et al. (2015), Li et al. (2018), Rusch and Mishra (2021), Smith et al. (2022), amongst others. Differently, sequential image generation is performed by predicting images in pixel-by-pixel manner (Katharopoulos et al., 2020). Image completions and unconditional samples are also usually considered. The former takes as input a portion of an image (that can be considered as a prompt, and it is usually approached by Transformers architectures (Katharopoulos et al., 2020)) and proceeds in generating the remainder of the image. The latter comprises the task of generating images with no condition in any context (such as a prompt text or another image). Ororbia et al. (2020) proposed a video-based generative modeling assessment, exploiting BouncingMNIST, BouncingNotMNIST and BouncingFashionMNIST in sequence prediction, zero-shot adaption and online continual learning setting. In BouncingMNIST, each sequence is a 20-frame video consisting of two randomly chosen digits from MNIST, moving and bouncing around in a 64×64 frame. Digits are placed at random initial locations within the overall patch, are assigned a velocity and move and bounce off edges of the overall frame. When digits occupy the same location, they occlude each other and overlap. In BouncingNotMNIST, fonts/glyphs from the NotMNIST dataset[24] are exploited, with the same characteristics (e.g., dimensions, sequence length, etc.) of BouncingNotMNIST. The same holds for BouncingFashionMNIST, with exemplars taken from the FashionMNIST dataset (Xiao et al., 2017). Sequence prediction is tackled both with an encoder/decoder approach (first 10 frames are provided as input to the encoder, while the decoder has to generate the next 10 frames) and by training on input sequences of 10 frames to reconstruct those 10 frames as well as predict 10 other future frames (Srivastava et al., 2015) (a recurrent architecture processes image features extracted by a convolutional model). In zero-shot adaptation, a model that is trained on one dataset (e.g., BouncingMNIST) is tested on its ability to generate samples from another set (e.g., BouncingNotMNIST). In the continual online learning scenario (Mai et al., 2022), as discussed in Ororbia et al. (2020), models are trained on a stream composed of three of the aforementioned datasets, with a single pass over the , using mini-batches of size 1 (i.e., pure online learning), which is a much more realistic setting when faced with infinite streams of patterns, where data arrives from different tasks at different times. The models are trained in a one-shot manner on a concatenation of the three datasets. We frame these approaches under the genMNIST identifier in Table 8.

CIFAR Similarly to the case of MNIST, CIFAR-10/100 (Krizhevsky et al., 2009) are also exploited as datasets for sequence-oriented models. A common approach is to flatten each CIFAR image (32×32 pixels with three channels) along height and width and process at each time step the three channels, resulting in $L = 1024$. Lim et al. (2023) apply random horizontal flipping before transforming it to a sequence with three channels by simply reshaping the signal. TLB (Didolkar et al., 2022) (see Section 3) tests the generalization abilities of the model by comparing its performance on images of higher resolution (128×128 pixels) with respect to the ones experienced during training. A ViT (Dosovitskiy et al., 2020) is exploited as the perceptual module and a Temporal Latent Bottleneck module is added to it. The input image is split into patches of 4×4 pixels and fed in raster order to the model. To predict the classification scores, the mean across the final Temporal Latent Bottleneck state vectors is computed and the resulting representation is processed through an MLP. Zhai et al. (2021) consider the problem of image autoregressive modeling, by minimizing the negative log likelihood. In this case, images are processed in scanline order resulting in an unrolled sequence length of $L = 3072$. Each sub-pixel is represented as a 256-way discrete variable. A different scenario involves the noise-padded CIFAR dataset (npCIFAR) as described in Chang et al. (2018). In this setup, images undergo row-wise processing and are flattened across channels, creating sequences with a length of $L = 32$. To introduce a distraction factor, the sequence is extended by additional uniform random numbers (for 968 steps) up to a sequence length of 1000. CIFAR10/100 dataset are considered in Gu et al. (2021b), Lim et al. (2023), Rusch and Mishra (2020, 2021), Smith et al. (2022), Zhai et al. (2021).

ImageNet Following similar ways of building sequences from static images, also the popular ImageNet dataset (Dosovitskiy et al., 2020) has been exploited in the literature of sequence processing. RFA (Choromanski et al., 2020b) exploits a 64×64 ImageNet variant, proposed by Parmar et al. (2018), processed pixel-by-pixel and considering each channel as independent, resulting in $L = 12288$. Other models simply integrate their approach into pre-existing architectures, such as HRGN (Qin et al., 2023c), which is injected into a DeiT (Touvron et al., 2021) structure and tested on the ImageNet-1k dataset for image classification. SGConv (Li et al., 2022b) replaces the 7×7 2D convolutional kernels of ConvNeXt with SGConvs, which treats 2D features as sequences. Additionally, Irie et al. (2022b) leverage the Mini-ImageNet dataset (Vinyals et al., 2016), composed of 100 classes with 600 examples each, typically resized to 84×84 pixels (Ravi & Larochelle, 2016). Works exploiting ImageNet are the following: Choromanski et al. (2020b), Irie et al. (2022b), Li et al. (2022b), Liu et al. (2022b), Poli et al. (2023), Qin et al. (2023c)

Action Recognition Video Activity Detection focuses on predicting actions at each time step within a video. Thus, unlike video classification, this task deals with long videos containing multiple overlapping activities, introducing a challenging problem due to the need to strongly exploit and build an expressive temporal context. The recent TTM (Ryoo et al., 2023) (see Section 3) evaluated causally masked inference, providing activity predictions for incoming frames without considering future steps, in the Charades dataset (Sigurdsson et al., 2016), which contains 9.8k videos showcasing 157 daily household activities, with 7.9k training and 1.8k validation clips. Videos can include multiple overlapping activities, requiring models to predict various activity classes per frame, taking into account interactions and longer temporal contexts. The average video length is 30 seconds, creating a challenging scenario for temporal activity detection. The AVA v2.2 (Gu et al., 2018) dataset is also employed to test spatio-temporal activity detection, and is composed of bounding box annotations of 80 atomic visual actions in 430 15-minute movie clips. Li et al. (2018) leverage the NTU RGB+D dataset (Shahroudy et al., 2016), which is a skeleton-based action recognition task, comprising 56880 sequences belonging to 60 action classes and each containing 20 frames.

7.3. Natural language benchmarks

Several architectures (both in the context of Linear Transformers and SSMs, e.g., TransNormer (Qin et al., 2023a), GLA (Yang et al., 2023), RWKV (Peng et al., 2023), Hyena (Poli et al., 2023), Mamba (Poli et al., 2023), Griffin (De et al., 2024), and others) are being exploited to setup LLMs, with the final goal of optimizing performances at scale. Thus, the tasks that are tackled to test such solutions are the ones characterizing the world of Natural Language Processing.

Language Modeling One of the peculiar characteristic behind the success of LLMs is their ability to model the generative likelihood of word sequences, i.e., they are able to predict the probabilities of future tokens. In other words, they can build powerful Language Models for a target language, being able to capture the syntax, grammar, and semantic properties. There exist several Language Modeling dataset, and many of the works described in this survey are evaluated in Language Modeling tasks. PennTreebank (PTB) (Marcus et al., 1993), WikiText-

[24] https://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html

103 (Merity et al., 2016) and The Pile (Gao et al., 2020) are commonly used for evaluating model performances terms of *perplexity*. PTB is a language model corpus containing 1 million words. Generally, models are trained to predict the next word (a variant denoted with PTB-w, evaluated with the perplexity metric) or character (referred to as PTB-c, evaluated with *bits-per-character* metric). The corpus contains $42,068$ sentences ($971,657$ words, average word-length of about 4.727 characters) of varying length (the range is from 3 to 84 words). The vocabulary for the character level models includes 49 unique symbols (including one for spaces). The standard train/valid/test split consists of 929k training words, 73k validation words, and 82k test words, with lowercase words, numbers replaced with token N, newlines replaced with <eos>, all other punctuation removed. The vocabulary is composed by the most frequent 10k words, with the rest of the tokens replaced by an <unk> token (Mikolov & Zweig, 2012). In literature, various segmentations of the corpus into training sequences of different lengths are being used ($L = 50$ for PTB-w in Li et al. (2018), Rusch et al. (2021), $L \in (70, 300)$ for PTB-w and $L = 150$ for PTB-c in Kag and Saligrama (2021b)). This task is considered in Bradbury et al. (2016), Dai et al. (2019), Kag and Saligrama (2021b), Kag et al. (2019), Kerg et al. (2019), Li et al. (2018), Ororbia et al. (2020), Rusch et al. (2021). WikiText-103 (Merity et al., 2016) is a long-range world-level language modeling benchmark. It contains 28k articles from English Wikipedia, with an average length of 3.6k tokens per article. In total, it contains 103M words. The preprocessing is more realistic with respect to PTB (e.g., vocabulary is not limited to 10k words, only words with a count below 3 are discarded). It is used in Bulatov et al. (2022), Dai et al. (2019), Gu et al. (2021a, 2020b), Huang et al. (2022), Irie et al. (2021), Kasai et al. (2021), Katsch (2023), Li et al. (2023b, 2022b), Ma et al. (2023), Mao (2022), Menick et al. (2020), Peng et al. (2021), Poli et al. (2023), Qin et al. (2022a,b, 2023c), Schlag et al. (2021), Wu et al. (2022a). The Pile dataset (Gao et al., 2020) is an 825 GiB English text corpus designed for training large-scale language models. It is composed of 22 diverse and high-quality datasets, including established Natural Language Processing datasets (PG-19, Books3, English Wikipedia, etc.) and 14 new high-quality curated language modeling datasets derived from heterogeneous sources (PubMed, ArXiv, GitHub, Stack Exchange, etc.–see Table 1 in Gao et al. (2020) for a full description)). Besides its impact in training large language models, The Pile can serve as a benchmark for testing cross-domain knowledge and generalization ability of language models. It is exploited in Bulatov et al. (2023), Gu and Dao (2023), Peng et al. (2023), Poli et al. (2023), Qin et al. (2023c), Sun et al. (2023a). Among the datasets contained in The Pile, the Project Gutenberg (PG-19) (Rae et al., 2019) consists of $28,602$ full-length books from the Project Gutenberg, published prior to 1919, hence representing distinct writing styles with respect to other datasets based on modern text. It contains $6,966,499$ English language words. When tokenized, each PG-19 book has between 50k-100k tokens. This datasets has been employed to test long range modeling capabilities of the approaches proposed in Hutchins et al. (2022), Pilault et al. (2023), Wu et al. (2022a). Datasets from Arxiv and GitHub were already been introduced in Wu et al. (2022b), prior to The Pile. The ArXiv dataset is a corpus of sources of technical papers from the "Mathematics" section, downloaded via the ArXiv Bulk Data Access. The GitHub dataset consists of source code extracted from diverse GitHub repositories featuring open-source licenses. In this dataset, all files within each GitHub repository are combined to form one long document. The evaluation metric commonly used with these datasets is bits-per-token (i.e., $\log_2$(perplexity), the lower the better). Used by Hutchins et al. (2022), Li et al. (2023b), Pilault et al. (2023). Enwik8 (Mahoney, 2011) is a character-level dataset consisting of 100M unprocessed bytes of the XML text dump of the English Wikipedia. Its vocabulary contains 204 characters, and it is exploited in Bulatov et al. (2022), Dai et al. (2019), Huang et al. (2022), Lei et al. (2018), Zhai et al. (2021).

Commonsense Reasoning A common experimental setting for evaluating LLMs consists in pre-training the model on The Pile and then testing the model capabilities on downstream zero-shot evaluation tasks. In particular, recent works leverage the Language Model Evaluation Harness (LM-Eval-Harness)[25] from EleutherAI (Gao et al., 2021), which mainly evaluates on tasks/datasets that measure common sense reasoning—in which the model should use "common sense" or world knowledge to perform inference. It is composed by several tasks: LAMBADA, HellaSwag, PIQA, ARC, WinoGrande. LAMBADA (Paperno et al., 2016) tests the modeling capacity on long-range contextual reasoning and language comprehension abilities. The model is given a certain context sentence, and it is asked to predict the last word of a target sentence based on the context. Accuracy and perplexity of the predicted last words are measured to assess the performance on language modeling tasks. The scaling law property (Hoffmann et al., 2022; Kaplan et al., 2020) is often measured, meaning that scaling language models would improve the accuracy and reduce the perplexity. LAMBADA contains novels from the Book Corpus (Zhu et al., 2015), and comprises $10,022$ passages, divided into $4,869$ development (that can be used to finetune models) and $5,153$ test passages (extracted from $1,331$ and $1,332$ disjoint novels, respectively). The average passage consists of 4.6 sentences in the context plus 1 target sentence, for a total length of ≈ 75 tokens. The dataset provides also training data, which include the full text of $2,662$ novels (disjoint from those in dev/test sets), comprising 203 million words. HellaSwag (Zellers et al., 2019)[26] expands the SWAG dataset (Zellers et al., 2018), which introduced the task of *commonsense natural language inference*: the model receives context from a video caption, along with four potential outcomes for the subsequent events. Among the provided choices, only one is correct, representing the actual succeeding caption in the video. Negatives are created using Adversarial Filtering, a method where a set of discriminators is employed to choose a tricky collection of incorrectly generated answers. According to the HellaSwag paper, models like BERT struggle with robust commonsense reasoning and tend to learn specific biases from the dataset. If there's a slight shift in language distribution, their performance drops significantly, even if the domain remains the same. HellaSwag introduces $70,000$ problems that are easy for humans (95.6% accuracy) but challenging for machines (50%). These problems consist of video captions from the ActivityNet Captions dataset and context and follow-up paragraphs from WikiHow. To assess how a model can adapt to novel scenarios, category labels sourced from WikiHow and ActivityNet are sampled to build 'zero-shot' evaluation sets. For each set, being it used for validation or testing, two subsets are generated. The first one includes $5,000$ 'in-domain' examples from categories encountered during training. The second subset comprises $5,000$ 'zero-shot' examples randomly selected from categories held out during training. In total, there are $70,000$ examples in the dataset. PIQA (Physical Interaction: Question Answering) (Bisk et al., 2020) benchmarks are about physical commonsense understanding. Given a "physical" goal ("*to separate egg whites using a water bottle…*") expressed in natural language and two possible solutions ((i) "*Squeeze the water bottle and press it against the yolk. Release, which creates suction and lifts the yolk;*" (ii) *"Place the water bottle and press it against the yolk. Keep pushing, which creates suction and lifts the yolk."*), a model must choose the most appropriate solution. The dataset assesses the capacity of natural language understanding models to connect text with a robust intuitive-physics model of the world. Humans effortlessly choose answer (i) because separating the egg involves pulling out the yolk, a task that machines find easily misleading. The dataset consists of more than $16,000$ training QA pairs with an additional $\approx$ 2k and $\approx$ 3k QA held out for development and testing, respectively. Goal sentences have an average length of 7.8 words, while correct/incorrect solutions have an average length of 21.3 words. This results in over 3.7 million lexical tokens within the training data. The AI2 Reasoning Challenge (ARC)

[25] https://github.com/EleutherAI/lm-evaluation-harness

[26] Short for Harder Endings, Longer contexts, and Lowshot Activities for Situations With Adversarial Generations.

(Clark et al., 2018) consists of a collection of 7787 natural grade-school science questions (typically 4-way multiple choice), authored for human standardized tests. It is partitioned into two sets: Challenge Set (ARC-c), which contains 2590 questions which are incorrectly answered by both a retrieval-based algorithm and a word co-occurrence algorithm. An Easy Set (ARC-easy) contrains the remainder of the questions. WinoGrande (Sakaguchi et al., 2021) is a large-scale dataset of 44k pronoun resolution problems. The task is to identify to which of two subject a *pronoun*[27] is referring to (e.g., "*Robert woke up at 9:00am while Samuel woke up at 6:00am, so _ had less time to get ready for school.*"; Options: Robert / Samuel). Generally, there is a twin sentence with nearly identical structure but where the answer is the opposite. It has been designed to be unsolvable for statistical models that rely on basic word associations. Going back to the main container of the just described datasets, the LM-Eval-Harness, we mentioned that it has been used in several recent works (De et al., 2024; Gu & Dao, 2023; Peng et al., 2023; Qin et al., 2023a; Sun et al., 2023a; Yang et al., 2023). Apart from LM-Eval-Harness, other comprehensive benchmarks for commons sense reasoning have been tested. Measuring Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2020) is a versatile benchmark for large-scale evaluation of multi-task knowledge understanding. It consists of 57 tasks including elementary mathematics, US history, computer science, law, etc. It collects $15,908$ questions in total, split into a few-shot development set, a validation set, and a test set. As shown in existing work, LLMs mostly outperform small models by a substantial margin on this benchmark, also showing scaling laws in model size. We point the reader attention towards a recent survey on LLMs (Zhao et al., 2023) for further details on other commonly used datasets, such as CMMLU (Li et al., 2023a), C-Eval (Huang et al., 2024) BoolQ (Clark et al., 2019), OpenBookQA (Mihaylov et al., 2018) and others.

Machine Translation Another sequence-oriented benchmark in the context of language generation is conditional text generation (Li et al., 2022a), which focuses on generating texts satisfying specific task demands based on the given conditions. A common task is machine translation (Bahdanau et al., 2014), whose goal is the translation of text or speech from a source language to another. Among the most common benchmarks, the WMT (several editions—2014 (Bojar et al., 2014) up to 2022 (Kocmi et al., 2022)) contains translation tasks from/to English to/from several languages, such as Czech, French, German, Hindi, and Russian. A common testbed is to use the WMT'14 English-to-German, which contains 4.6M sentence pairs, and the WMT'17 the Chinese-to-English split which is composed by 20.6M sentence pairs. The case-sensitive NIST BLEU score (Papineni et al., 2002) is commonly adopted as the evaluation metric. It has been tackled in Chen et al. (2019, 2018a), Hao et al. (2019), Kasai et al. (2021), Ma et al. (2023), Peng et al. (2021), Schlag et al. (2021). A similar benchmark is the IWSLT German-English spoken-domain translation (Cettolo et al., 2014). It consists of $209,772$ sentence pairs from transcribed TED and TEDx presentations, with a mean sentence length of 103 characters for German and 93 for English. It has been considered in Bradbury et al. (2016), Peng et al. (2021).

Other Language-oriented Benchmarks Apart from the generation of high-quality natural language text, several models show strong abilities to generate regular and formal languages, such as computer programs (i.e., code synthesis) (Huang et al., 2022; Zhao et al., 2023). Another popular direction for evaluation is the one is sentiment classification. The principal testbed is the IMDb movie review dataset (Maas et al., 2011), a collection of 50k movie reviews, equally balanced into positive and negative reviews and processed into equal-size train and test sets. Sequence length range from hundreds to thousands, with an average document length of 231 words. The aim of this binary sentiment classification task is to decide whether a movie review is positive or negative. Several works considered this task (Bradbury et al., 2016; Gu et al., 2020a; Qin et al., 2022b; Rusch & Mishra, 2020, 2021).

DNA Modeling DNA can be easily connected to language, given that it consists of a sequence of discrete tokens (i.e., nucleotides) belonging to a limited vocabulary. Indeed, multiple LLM-based solutions for genomics have been proposed. Additionally, recent works have highlighted the advantages of by considering long-range interaction when predicting gene expression (Avsec et al., 2021). Recent works in the context of this survey (Gu & Dao, 2023; Nguyen et al., 2024) explore scaling laws across model size and sequence length as well as downstream classification. Pretraining is performed via causal language modeling (next token prediction) on the Human Genome HG38 dataset, which consists of a single human genome containing 4.5 billion tokens (DNA base pairs) in the training split. Please refer to the cited papers for further details.

7.4. Speech/audio benchmarks

Processing sequences of "raw" audio data can be challenging for neural models, due to high-frequency sampling, resulting in very long sequences. Some traditional approaches involve complex pipelines that require extracting mixed-and-matched hand-crafted features. The Speech Commands (Warden, 2018) dataset provides 1-second raw audio waveforms sampled at 16000Hz (i.e., a 1-D sequence of $L = 16000$), consisting of $105,829$ recordings from $2,618$ speakers – both background noise and a vocabulary of 35 spoken words (such as 'left', 'right', etc.). The task is to classify which word was spoken. While several works (Kag et al., 2019; Kidger et al., 2020) exploit a pre-processed variant with standard mel-frequency cepstrum coefficients (MFCC), recent models tackle the problem directly processing the raw signal (Gu et al., 2022, 2021a,b; Gupta et al., 2022a; Smith et al., 2022). Other recent works (Gu & Dao, 2023) explits a subset of the dataset (referred to as SC09) consisting of spoken digits "zero" through "nine" ($31,158$ training utterances—8.7 hours in total—-by $2,032$ speakers). This is usually tackled as an autoregressive speech generation problem. Other similar dataset are: TIMIT (Garofolo et al., 1993), leveraged in Erichson et al. (2021); Beethoven (with the setting described by Mehri et al. (2016)) consists of recordings of Beethoven's 32 piano sonata; YouTubeMix is a 4 h dataset of piano music. The latter two dataset are considered in Goel et al. (2022).

7.5. Time series benchmarks

Time series consists of sequential samples paired with time-stamps, generally (but not necessarily) sampled at equally spaced points in time. Such data is often characterized by very long range recurring patterns, that can be exploited to extract meaningful statistics. Several problems can be faced with these data, such as (i) the classification of a time series based on its characteristics; (ii) regression/prediction analysis to test relationships between one or more different time series; (iii) time series forecasting, searching for a model capable of predicting future values of the series based on previously observed ones.

Classification The Eigenworms dataset (Brown et al., 2013) is a collection of 259 very long sequences, i.e., $L = 17,984$, describing the motion of a particular worm species. The kind of movement performed by these worms is a useful indicator for understanding behavioural genetics. In particular, these worms adopt certain shapes when placed on an agar plate, that can be represented by combinations of six base shapes, referred to as *eigenworms*. The time series is constituted at each time step by six scalars representing the amplitudes along each dimension when the shape is projected onto the six eigenworms. The goal is to classify a worm as either wild-type or one among four mutant types, and has been tackled in Irie et al. (2022a), Lim et al. (2023), Rusch and Mishra (2021). The PhysioNet Sepsis (Reyna et al., 2020) is an healthcare task, composed by an irregularly sampled time series with partially missing features. It consists in a binary prediction problem of

[27] In practice it is formatted as a fill-in-the-blank problem, where the blank corresponds to the mention of one of the two names in the context.

sepsis, using a time series comprising 34 medical features recorded during patients' stays in an ICU. Each sequence is accompanied by five static patient features (e.g., age). The sequences are short, with less than 72 frames, and the data points are irregularly sampled with many missing entries, posing a challenge. There are two versions of the task: one with observation intensity information (OI) and one without (no-OI), which includes an extra input feature indicating the time stamp of each observation to provide information on measurement frequency. Tested in Irie et al. (2022a), Kidger et al. (2020), Morrill et al. (2021). Human activity recognition is performed in the HAR-2 (Anguita et al., 2012) dataset, a collection of tracked human activities measured by an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. Six activities were binarized to obtain two merged classes (Sitting, Laying, Walking_Upstairs) and (Standing, Walking, Walking_Downstairs). HAR-2 contains $\approx$ 7k training sequence each of them composed by 128 time steps represented with $\approx$ 1k features. Exploited in Kag et al. (2019), Rusch and Mishra (2020).

Regression/Prediction The BDIMC healthcare datasets is composed by $7,949$ sequences, and it is aimed at predicting three vital signs of a patient. It is part of the TSR archive (Tan et al., 2020), presenting clinical data from the Beth Israel Deaconess Medical Center. PPG and ECG signals were sampled with a frequency of 125Hz for 8 minutes each. The resulting two-dimensional sequences have a length of $L = 4000$. The goal is to predict a patient's respiratory rate (RR), heart rate (HR), and oxygen saturation (SpO2), based on PPG and ECG signals. RMSE is exploited to measure performances (Gu et al., 2021b; Rusch & Mishra, 2021).

The Pendulum benchmark (Becker et al., 2019; Schirmer et al., 2022) has been recently used Smith et al. (2022) to investigate models ability to handle observations received at irregular intervals. It is composed by a $L = 50$ long sequence of images having a resolution of 24×24 pixels, corrupted with a correlated noise process and sampled at irregular intervals from a continuous trajectory. The model is tasked to predict the sine and cosine of the angle of the pendulum, which follows a nonlinear dynamical system, without any explicit information on the velocity.

Forecasting The Mackey-Glass (MG) dataset (Mackey & Glass, 1977) tests the ability of a network to model chaotic dynamical systems. A sequence of one-dimensional observations–generated by solving the MG differential equations–are streamed as input, and the network is tasked with predicting the next 15 values in the sequence, with an MG time-constant of 17 steps. It is used in Gu et al. (2020a), Voelker et al. (2019). In the existing literature, the prediction of long sequence time-series, such as electricity consumption planning, is usually referred to as Long sequence time-series forecasting (LSTF). Among the most known datasets, the Electricity Transformer Dataset (ETD) [28] contains the oil temperature and 6 power load features of electricity transformers from different regions of a province of China, over a period of two years. The ETT-small split is composed by two dataset coming from two electricity transformers located in different regions, where data points have been recorded every minute (ETTm1, ETTm2) or hour (ETTh1, ETTh2). Each dataset contains $70,080$ data points with 8 features, including the date, the predictive value "oil temperature", and 6 different external power load features. Electricity Consuming Load (ECL)[29] is another benchmark which collects the electricity consumption of 321 clients from 00:00 of January 1st 2012 to January 31st 2014, registered every 15 min. Each column represents the energy consumption of a client over this period. Weather[30] records 11 climate features in time intervals of one hour from January 1st 2010 to December 31st 2013. Since ETT, ECL and Weather are obtained through real-world monitoring operations (via sensors), we will refer to them as SensorData in Table 8, and remark that they have been exploited in several works in the context of this survey (Gu et al., 2021a; Huang et al., 2022; Zhang et al., 2023).

[28] https://github.com/zhouhaoyi/ETDataset
[29] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014
[30] https://www.ncei.noaa.gov/data/local-climatological-data/

7.6. Reinforcement learning

Sequential models are also sparingly tested for sequential decision making in reinforcement learning tasks. There is not a unified testbed in this setting. Didolkar et al. (2022) evaluate the BabyAI (Chevalier-Boisvert et al., 2018) benchmark, which provides a suite of environments where the agent has to carry out a given instruction (e.g., going to an object, placing an object beside another one, opening a door with a key, etc.) in a partially-observable maze. The goal is to produce an autoregressive generative model that predicts actions conditioned on the past context. Also the Atari benchmark (Chen et al., 2021) is frequently considered, exploiting a causal mask and supervised training, to match the actions in the offline dataset conditioned on the future expected returns and the past history. Other heterogeneous reinforcement learning benchmarks are evaluated in Gu et al. (2020b), Irie et al. (2022b), Pramanik et al. (2023).

7.7. Long range arena

The previous sections highlighted the absence of an established common set of benchmarks to evaluate sequential models that handle long-range tasks. Indeed, when comparing the experimental setup of the previously described papers, it can be observed that there is not a consensus on a common testbed. Models are evaluated on heterogeneous tasks and datasets ; even when using the same dataset, different works often preprocess it in a different manner, making the comparison of their performances very difficult. Moreover, many of the benchmarks do not directly target long-range modeling ability. Finally, recent works fall short in decoupling the effect of large-scale pretraining from the inductive bias arising from a certain data distribution.

In an effort to standardize long-range models evaluation, Tay et al. (2020) proposed the Long Range Arena (LRA) benchmark suite. LRA includes synthetic and real-world tasks to assess the ability of sequential architectures to reason in such scenarios, considering different types of data and conditions. It focuses on understanding how well architectures can handle long sequences having certain hierarchical or spatial structures, and aims to compare their performance across various settings. In details, it consists of 6 tasks: (*i*) Long ListOps, (*ii*) Text (*iii*) Retrieval (*iv*) Image (*v*) Pathfinder and its extreme long version, (*vi*) Path-X. Sequences constituting these task have a length $L \in [1k, 16k]$, encompassing modalities and objectives that require similarity, structural, and visuospatial reasoning. The LRA suite is becoming increasingly popular in the scientific literature and has been exploited by several recent works mentioned in the context of this survey Gu et al. (2022, 2021a), Gupta et al. (2022a,b), Hasani et al. (2022b), Li et al. (2022b), Liu et al. (2022b), Ma et al. (2023), Orvieto et al. (2023), Peng et al. (2021), Qin et al. (2022a,b, 2023c), Smith et al. (2022), Zucchet et al. (2023b).

Long ListOps Long ListOps is a longer version of the ListOps task proposed in Nangia and Bowman (2018), aimed at assessing the model ability to process hierarchically structured data in a long-context scenario. In each sequence, some mathematical operators (e.g., max, mean, median and sum_mod) and integer operands (range 0 to 9) are enclosed by delimiters (brackets) in prefix notation. The goal is to compute the integer result of the mathematical expression, as exemplified by the following (short) sequence,

```
INPUT: [MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]
OUTPUT: 5
```

Input symbols are encoded via one-hot vectors, having 17 unique possible configurations (opening brackets and operators are grouped into a single token). Long ListOps tests the ability to reason hierarchically while handling long contexts with varying length, up to $L = 2$k, with a reserved end-of-sequence token. The final goal is ten-way classification (the integer result of the expression). There are 96k training sequences, 2k validation sequences, and 2k test sequences.

Text Text is a byte-level text classification task, based upon the IMDb binary sentiment classification previously described (i.e., classify

Fig. 19. Negative (left) and positive (right) samples from Pathfinder.

whether a movie review is positive/negative). Text sequences are processed at byte/character-level (characters are encoded as one-hot vectors, with 129 unique values possible), with a fixed max length of $L = 4$k, and truncated or padded when necessary. The evaluation metric is the accuracy. Compared to ListOps, the model needs to reason with compositional unsegmented real-world data, with less defined boundaries, that must be learnt from data. Differently from character-level language modeling, where the local context is sufficient to infer the next character, here the model is required to grasp the overall sentiment of the processed text, hence it needs to compose characters into words and then higher-level concepts. There are 25k training examples and 25k test examples, while no validation set is provided.

Retrieval Retrieval can be used to assess the model capability of encoding long sequences into representations suitable for similarity-based matching. Couples of texts are compressed and concatenated, then provided to a linear classifier. The text is sampled from the ACL Anthology Network (Radev et al., 2013) dataset. Given two textual citations, where characters are encoded as a sequence of integer tokens (one-hot vector with 97 unique values), the task is to classify whether the two citations are equivalent. Byte/character level processing is performed, resulting in sequence length of $L = 4$k for each document (i.e., total text length 8k for this task). The performances of this binary classification task are measured with accuracy. There are $147,086$ training pairs, $18,090$ validation pairs, and $17,437$ test pairs.

Image The Image task corresponds to the already introduced sCIFAR, an image classification task on sequences of pixels, described in Section 7.2. A 32×32 pixels image is flattened in raster scan into a sequence of 1024 pixels. The goal is to classify (10-way classification) the image by processing the 1D pixel sequence. Models are required to capture 2D spatial hierarchical structure between pixels even if they are processed sequentially. Images belong to the CIFAR-10 (Krizhevsky et al., 2009) dataset, and are processed in a single gray-scale channel (each pixel represented with 8-bit pixel intensity, hence vocabulary size of 256).[31] There are 45k training examples, 5k validation examples, and 10k test examples.

Pathfinder Pathfinder is a synthetic visual task originally introduced in Kim* et al. (2020), Linsley et al. (2018) for learning long-range spatial dependencies.

A 32×32 grayscale image contains two points (a start and an end point, represented as small circles—see Fig. 19). There are dashed lines over the image. The task is to make a binary decision whether the two points are connected by a path (Fig. 19-left) or not (Fig. 19-right). Models process the image pixel sequence (raster scan), thus $L = 1024$. There are 160k training examples, 20k validation examples, and 20k test examples.

Path-X Finally, Path-X is an harder version of Pathfinder composed by 128×128 images, resulting in sequences of $L = 16$k.

7.8. Empirical comparison: transformers vs. SSMs

We now provide an empirical comparison between attention-based Transformers and SSMs across a range of synthetic and real-world sequence modeling tasks. By evaluating performance on memory-intensive benchmarks like the Copy task, Induction Heads, Associative Recall, and LRA, the analysis highlights where each architecture excels or falls short, especially as sequence lengths grow or memory constraints tighten.

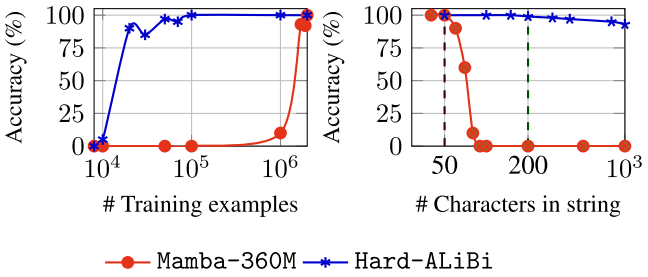

Fig. 20. Revisited from Jelassi et al. (2024). (Left) The authors train models to copy strings of length ≤ 300 and evaluate string-level accuracy on strings of length 300. Transformers train much faster than SSMs. Indeed, the Mamba-360M needs two orders of magnitude more w.r.t. Hard-ALiBi Transformer. (Right) The authors train models to copy on strings of length ≤ 50 until all models are perfect in-distribution and evaluate string-level accuracy. Purple dashed line indicates maximum training string length and green dashed line indicates context window during training. Evaluating on longer inputs, the transformer models dramatically outperform the SSMs..

Copy task The synthetic Copy task (Hochreiter & Schmidhuber, 1997) remains the most direct probe of a model's raw memory capacity: the network is asked to reproduce an input string verbatim after a sentinel token. Recent results demonstrate a decisive advantage for attention-based Transformers over SSMs on both data-efficiency and length generalisation. Fig. 20, revisited from Jelassi et al. (2024), contrasts the strongest publicly reported results for Transformers and SSMs. Hard-ALiBi (Jelassi et al., 2024), a Transformer architecture exploiting positional encodings that bias query-key attention scores with a penalty proportional to their distance, reaches $\geq 95\,\%$ string-level accuracy on length-300 sequences after only 1×10^4 training examples; conversely, Mamba-360M needs two orders of magnitude more (Fig. 20–left).[32] When evaluated on out-of-distribution lengths (train on ≤ 50, test on $300 / 1000$), the Transformer preserves $\geq 90\,\%$ accuracy, while Mamba and other gated SSMs collapse to chance (Fig. 20–right).

Induction Heads and Associative Recall Synthetic memory benchmarks such as Induction Heads (IH) and Associative Recall (AR) provide a minimal yet revealing playground for analysing how sequence models store and retrieve information. Because the training context is kept short while the evaluation context can reach millions of tokens, these tasks expose the limits of each architecture's memory mechanism. We report in Table 6 the results in the original papers (Dao et al., 2023; Gu & Dao, 2023). All models are two-layer variants trained on length-256 sequences for 2×10^5 updates and evaluated on powers of two up to $L = 1,048,576$ tokens. These tasks expose three qualitatively different memory regimes that map cleanly onto the architectures in Table 6. Plain linear SSMs such as S4D and GSS compress the entire past into a fixed-width latent state. As the number of distinct pairs approaches that width the representation collides, which explains the sharp drop to $\approx 35\,\%$ accuracy on IH even for 256–token contexts. H3 augments the recurrent path with a learned *shift* channel so that every incoming token is explicitly available in the next state's slot, and combines this with multiplicative gating. The resulting mechanism behaves like a differentiable hash table that stores each token exactly once while using the gate for equality tests. This prior yields $99.8\,\%$ accuracy on AR and perfect IH, with the remaining $0.2\,\%$ error attributable to rare collisions when two identical keys land in the same step. Mamba replaces fixed convolutions

[31] Some recent works process the coloured images instead (Orvieto et al., 2023).

[32] The authors use their standard 300-token setup to keep compute and optimisation schedules identical across models.

Table 6
Sequence-level accuracy on Induction Heads (IH) and Associative Recall (AR)–results taken from Dao et al. (2023), Gu and Dao (2023).

Model	Parameters	IH	AR
Transformer (Vaswani et al., 2017)	0.10M	100 %	100 %
S4D (Gu et al., 2021b)	0.10M	35.6 %	86.0 %
GSS (Mehta et al., 2022)	0.10M	6.8 %	78.0 %
H3 (Dao et al., 2023)	0.15M	100 %	99.8 %
Mamba (Gu & Dao, 2023)	0.07M	100 %	100 %

Table 7
Exact test accuracy on the six Long-Range Arena tasks. Results are taken from table of the original papers (Orvieto et al., 2023; Smith et al., 2022).

Task (length)	Trans.	Mega	S5	LRU
ListOps (2 k)	36.37 %	63.14 %	62.15 %	60.2 %
Text (4 k)	64.27 %	90.43 %	89.31 %	89.4 %
Retrieval (4 k)	57.46 %	91.25 %	91.40 %	89.9 %
sCIFAR-10 (1 k)	42.44 %	90.44 %	88.00 %	89.0 %
Pathfinder (1 k)	71.40 %	96.01 %	95.33 %	95.1 %
Path-X (16 k)	7.00 %	97.98 %	98.58 %	94.2 %

with input–dependent transition matrices. By deciding *at runtime* which information to propagate, Mamba mimics the selective read–write behaviour of attention yet keeps both time and memory linear in L. Consequently it sustains 100 % accuracy from 64 up to 1,048,576 tokens. When GPU memory allows, Transformers remain the simplest drop–in solution and excel at many-to-many interactions. H3 and especially Mamba become preferable for million-token contexts where linear memory is a hard requirement.

Long Range Arena The LRA benchmark (Tay et al., 2020) comprises six tasks with sequence lengths ranging from 1,024 to 16,384 tokens, as described above. We compare the classical Transformer baseline against different SSMs architectures, reporting results from Orvieto et al. (2023), Smith et al. (2022) in Table 7.

Three main trends emerge from Table 7. Mega, which augments a standard self-attention block with a diagonal state-space kernel, delivers the top score on four of the six tasks–peaking at 91.25 % on the document-pair Retrieval task and 96.0 % on Pathfinder. The classical Transformer, by contrast, collapses to chance-level accuracy (7 %) on the 16k-token Path-X and lags by 20–50 pp everywhere else. S5 dominates the very long regime. When sequence length jumps to 16k, the continuous-time SSM of S5 reaches 98.58 % on Path-X. It also ties or edges Mega on the image and Pathfinder tasks while keeping $\mathcal{O}(L)$ memory and compute. LRU shows how far a linear RNN can go. With a purely recurrent core, stable-exponential parameterisation and a single γ-norm, LRU matches S5 within one percentage point on five tasks and even surpasses it on sCIFAR-10 (89.0 % versus 88.0 %). Taken together, these results suggest a clear trade-off. If memory footprint is no object, the Mega transformer still gives the best average accuracy. For workloads dominated by $\geq 10\,000$-token sequences or tight latency budgets, however, S5 and the much simpler LRU offer comparable accuracy with strictly linear scaling and constant-time autoregressive generation.

In summary, Transformers outperform SSMs on data efficiency and generalization in the Copy task, requiring far fewer samples and maintaining high accuracy on longer sequences. In synthetic memory tests (IH, AR), advanced SSMs like H3 and Mamba match Transformer performance while scaling linearly with sequence length. On LRA benchmarks, Mega leads in accuracy, but SSMs like S5 and LRU offer competitive results with better efficiency, especially on very long sequences. Overall, Transformers are ideal when resources allow, while modern SSMs are better suited for long-context, resource-constrained settings.

8. Discussion and future directions

In this Section, we analyze some open problems and issues that have been recently pointed out in the works described in this survey, as well as highlighting possible future avenues for research. Despite the individual strengths of the architectures surveyed in previous sections, it is now increasingly evident that no single paradigm–be it pure Transformer, RNN, or SSM–can fully address the challenges posed by long-sequence modeling. Transformers offer excellent performance and global receptive fields, but suffer from quadratic complexity and require full sequence availability. RNNs and SSMs, on the other hand, offer efficient online processing and constant memory per time step, but are limited in terms of expressivity. These complementary strengths and weaknesses have naturally motivated the exploration of *hybrid* models that aim to combine the parallelism and global context handling of Transformers with the recurrence and efficiency of RNNs and SSMs. This section builds on this perspective, outlining both open issues and emerging design trends that point toward unified architectures capable of overcoming the fundamental trade-offs of current solutions.

8.1. Limitations of transformers and recurrent models

The previous sections explored recent progress in Transformers and SSMs, highlighting the resurgence of recurrent mechanisms. Future research directions should be informed by an understanding of the current limitations of these model classes. There is potential for trade-offs and hybrid approaches, alongside a need to co-design models with the computational hardware they target.

Transformers A series of recent studies have evaluated Transformers across various tasks to uncover their limitations. Hahn (2020) demonstrated that Transformers struggle with recognizing simple patterns, such as parity or properly nested parentheses, when faced with long inputs. Peng et al. (2024b) further investigated these limitations, focusing on function composition. For instance, tasks involving multi-step reasoning over several facts remain challenging for Transformers. Although chain-of-thought prompting can alleviate some issues, it substantially increases the number of required tokens. Furthermore, Transformers have shown shortcomings in time series forecasting tasks, where simpler architectures have outperformed them Zeng et al. (2023). Early theoretical work established that, under idealized assumptions such as infinite-precision arithmetic and arbitrarily powerful feed-forward subnets, transformers are “Turing complete” and can simulate a general-purpose machine (Pérez et al., 2019). However, once we impose realistic constraints–most critically, limiting attention to hard (discrete) patterns–their power collapses into well-studied Boolean circuit classes. Boolean circuits are acyclic networks of logical gates (e.g., AND, OR, NOT) whose depth and size determine how “parallel” or “sequential” a computation can be. Using circuit-complexity techniques, Hahn (2020) and Hao et al. (2022) showed that transformers with hard attention correspond to non-uniform AC^0 circuits, which cannot even compute the majority function on n bits. Allowing softer attention extends them to L-uniform TC^0 (the class of constant-depth, polynomial-size threshold circuits constructible in logarithmic space–informally, it can be thought of as the class of problems that can be solved with extremely parallel constant-depth computation), but Merrill and Sabharwal (2023a,b) demonstrated that even this model cannot express inherently sequential computations. In particular, NC^1-hard state-tracking problems–such as permutation composition–fall outside TC^0 and thus lie beyond the reach of any realistic transformer model (Liu et al., 2022a). Permutation composition underpins fundamental tasks like playing chess, evaluating code, or tracking entities in a narrative; since these problems require carrying and updating an internal state step by step, transformers constrained by finite-precision or limited attention cannot solve them in the worst case.

Recurrent models Despite the computational advantages of recurrence-based models, performances are not always on par with re-

Table 8
Sequence processing benchmarks. A list of the most common benchmarks (first column) and papers exploiting them (other columns). Each column report papers described in a different sections of this survey.

	Benchmarks	Transformers embracing Recurrence (Section 3)	Deep State-Space models (Section 4)	Novel RNNs architectures (Sections 5–6)
Synthetic	Adding Problem (Hochreiter & Schmidhuber, 1997)	(Arjovsky et al., 2016; Le et al., 2015)	–	(Gu et al., 2020b; Kag & Saligrama, 2021b; Li et al., 2018; Rusch & Mishra, 2020; Rusch et al., 2021)
	Copy Task (Hochreiter & Schmidhuber, 1997)	(Arjovsky et al., 2016; Ba et al., 2016b; Bulatov et al., 2022; Didolkar et al., 2022; Menick et al., 2020)	(De et al., 2024; Gu & Dao, 2023; Gu et al., 2020a; Zucchet et al., 2023b)	(Gu et al., 2020b; Irie et al., 2023b; Kerg et al., 2019; Menick et al., 2020; Park et al., 2023)
	Induction Heads		(De et al., 2024; Gu & Dao, 2023)	–
	Associative Recall	(Bulatov et al., 2022; Peng et al., 2024a; Yang et al., 2023)	(Dao et al., 2023; Gu & Dao, 2023; Poli et al., 2023)	(Beck et al., 2024; Liao et al., 2018)
Computer Vision	sMNIST	(Gu et al., 2020a, 2021a,b; Smith et al., 2022; Voelker et al., 2019)	(Erichson et al., 2021; Gu et al., 2020b; Kerg et al., 2019; Lim et al., 2021; Rusch & Mishra, 2020; Rusch et al., 2021)	(Kag & Saligrama, 2021b; Ororbia & Mali, 2019)
	psMNIST			(Erichson et al., 2021; Kag & Saligrama, 2021b; Park et al., 2023; Rusch & Mishra, 2020, 2021; Rusch et al., 2021)
	genMNIST	(Katharopoulos et al., 2020)		(Ororbia et al., 2020)
	CIFAR			(Gu et al., 2020b; Kag & Saligrama, 2021b; Lim et al., 2023)
	npCIFAR			(Rusch & Mishra, 2020, 2021; Rusch et al., 2021)
	ImageNet	(Choromanski et al., 2020b; Irie et al., 2022b; Liu et al., 2022b)	(Li et al., 2022b; Poli et al., 2023)	(Qin et al., 2023c)
Natural Language	WikiText103 (Merity et al., 2016)	(Bulatov et al., 2022; Dai et al., 2019; Huang et al., 2022; Irie et al., 2021; Kasai et al., 2021; Katsch, 2023; Li et al., 2023b; Mao, 2022; Peng et al., 2021; Qin et al., 2022a,b, 2023c; Schlag et al., 2021; Wu et al., 2022a)	(Gu et al., 2021a; Li et al., 2022b; Ma et al., 2023; Poli et al., 2023)	(Beck et al., 2024; Gu et al., 2020b; Menick et al., 2020)
	PTB(Marcus et al., 1993)	(Dai et al., 2019)		(Bradbury et al., 2016; Kag & Saligrama, 2021b; Kag et al., 2019; Kerg et al., 2019; Li et al., 2018; Rusch et al., 2021)
	The Pile(Gao et al., 2020)	(Bulatov et al., 2023; Peng et al., 2023; Qin et al., 2023c; Sun et al., 2023a)	(Gu & Dao, 2023; Poli et al., 2023)	
	PG-19 (Rae et al., 2019)	(Hutchins et al., 2022; Peng et al., 2024a; Pilault et al., 2023; Wu et al., 2022a)		
	Enwik8 (Mahoney, 2011)	(Bulatov et al., 2022; Dai et al., 2019; Huang et al., 2022; Zhai et al., 2021)		(Lei et al., 2018)
	ArXiv & GitHub (Wu et al., 2022b)	(Hutchins et al., 2022; Li et al., 2023b; Pilault et al., 2023)		
	LM-Eval-Harness (Gao et al., 2021)	(Peng et al., 2023, 2024a; Qin et al., 2023a; Sun et al., 2023a; Yang et al., 2023)	(De et al., 2024; Gu & Dao, 2023)	(Beck et al., 2024)
	WMT(Bojar et al., 2014; Kocmi et al., 2022)	(Chen et al., 2019, 2018a; Hao et al., 2019; Kasai et al., 2021; Peng et al., 2021; Schlag et al., 2021)	(Ma et al., 2023)	
	IWSLT(Cettolo et al., 2014)	(Peng et al., 2021)		(Bradbury et al., 2016)
	IMDb(Maas et al., 2011)	(Qin et al., 2022b)	(Gu et al., 2020a)	(Bradbury et al., 2016; Rusch & Mishra, 2020, 2021)
Time Series	Eigenworms (Brown et al., 2013)			(Irie et al., 2022a; Lim et al., 2023; Rusch & Mishra, 2021)
	HAR-2 (Anguita et al., 2012)		(Kag et al., 2019; Rusch & Mishra, 2020)	
	Mackey-Glass (Mackey & Glass, 1977)		(Gu et al., 2020a; Voelker et al., 2019)	
	BDIMC(Tan et al., 2020)		(Gu et al., 2021b)	(Rusch & Mishra, 2021)
	SensorData	(Huang et al., 2022)	(Gu et al., 2021b; Zhang et al., 2023)	
	Speech Commands (Warden, 2018)		(Gu et al., 2022, 2021a,b; Gupta et al., 2022a; Smith et al., 2022)	(Kag et al., 2019; Kidger et al., 2020; Rusch et al., 2021)
	Long Range Arena (Tay et al., 2020)	(Liu et al., 2022b; Peng et al., 2021; Qin et al., 2022a,b)	(Gu et al., 2022, 2021a; Gupta et al., 2022a,b; Hasani et al., 2022b; Li et al., 2022b; Ma et al., 2023; Orvieto et al., 2023; Smith et al., 2022; Zucchet et al., 2023b)	(Beck et al., 2024; Qin et al., 2023c)

spect to vanilla attention when considering specific tasks, also in synthetic settings (Dao et al., 2023; Jelassi et al., 2024). Indeed, the capability of recurrent models to effectively solve a target task is strongly conditioned upon how efficiently the processed sequence is *compressed* into a finite-size state (Gu & Dao, 2023), which acts as an information bottleneck.

The role of *gating* mechanisms turns out to be fundamental, given their influence on the model ability to control, in a context/input dependent manner, how information propagates or interacts along the temporal dimension. In fact, the relevance of an appropriate gating mechanism can be appreciated when considering that such a mechanism is present in many high-impact works described in this survey (Irie et al., 2021; Jing et al., 2019; Katsch, 2023; Peng et al., 2023; Schlag et al., 2021; Sun et al., 2023a; Yang et al., 2023; Zucchet et al., 2023a), going well beyond the original case of LSTMs. Indeed, the Mamba model (Gu & Dao, 2023) shows how a *selection* mechanism (i.e., a term which refers to the mechanistic action of a model to select or ignore inputs and facilitate data interaction along the sequence length, inspired by the original gating in Hochreiter and Schmidhuber (1997)) improves the capability of State-Space Models to solve tasks such as Selective copy or Induction Heads (see Section 7). In these task, content-aware reasoning is fundamental to be able to memorize relevant tokens and filter out the irrelevant ones. While the Linear Time Invariant (LTI) nature of previous SSMs (LSSL, S4 (Gu et al., 2021a,b)) allows to achieve efficient parallelizable solutions, their dynamics do not easily allow to select the correct information from the current context, making it harder to encode the information stored in the hidden state. Conversely, an appropriate selection mechanism introduces context-awareness, allowing the model to focus on or to filter out information. This element is also interesting in light of recent studies (see Section 5) suggesting that certain RNNs (equipped with linear recurrent layers interconnected by feedforward paths with multiplicative gating) might be implementing attention/selection "under the hood", when trained to solve simple in-context learning tasks (Zucchet et al., 2023a).

However, despite such advantages brought by gating/selection mechanisms, the performance of recurrence-based solutions still falls short of fully closing the gap with attention-based models. Several recent works investigated such a recurrence *expressivity gap* (Dao et al., 2023; Jelassi et al., 2024) with respect to attention. Indeed, inference in Transformers leverages the entire sequence processed so far, which, in autoregressive models, is stored in the KV-cache (see Section 3), to produce the final output. By definition, the memory usage of this approach increases over time, which becomes extremely expensive. At the cost of inefficiency, the attention mechanism possesses properties that standard stateful models are not able to embrace: explicitly recalling earlier tokens in the sequence (which influences their ability to copy from the input sequence, possibly by multiplying the attention scores with V) and comparing tokens across the sequence (which is achievable thanks to the quadratic attention matrix QK')). Jelassi et al. (2024) exploit the copy task (see Section 7) to theoretically and empirically prove this property. In short, they show that from the theoretical point of view a simple Transformer model could be capable (when empowered by proper positional encodings and under some other mild assumptions) of copying strings of length that are exponential in the number of heads of the Transformer. This result relies on the ability of the Transformer to implement a mechanism of "storage" and retrieval of sequences of n tokens (n-grams). Noticeably, the number of parameters depends only logarithmically on the input sequence length. The authors show that, conversely, any state-space model fails to solve the copy task, unless its latent state grows linearly with the sequence length. That is, SSMs cannot accurately copy strings with more bits than the size of the latent state.

Such theoretical findings have been confirmed empirically. Indeed, Transformers are both much more efficient at learning to copy (indeed, they need $\approx 10^4$ training examples to learn the task, whereas Mamba (Gu & Dao, 2023) needs more than $\approx 10^6$ samples, as mentioned in Section 7.8) and also generalize better to longer inputs (when trained on strings $L = 50$, they generalize to sequences with $L > 1000$, whilst Mamba fails to generalize out of the trained context). This finding confirm the theoretical results, i.e., to be able to copy the entire input sequence, stateful models need to fully store it in its state space, which requires the memory to grow linearly with the sequence length. These results hold even when considering pre-trained models, both in copy tasks and retrieval benchmarks (e.g., Phonebook Lookup, see Section 7).

These results are interesting in light of the fact that the *selection mechanism* underlying Mamba (see Section 4) was introduced specifically to be able to selectively propagate or forget information along the temporal dimension, depending on the current token. Similar intuitions have previously inspired the work of Dao et al. (2023), who proposed the H3 model. As we detailed in Section 4, their intuition on reducing the attention-recurrence expressivity gap is to empower the copy and retrieval ability of SSMs by (i) adding a layer with a shift matrix, that allows the state to copy from the input and pass that information to the next state, (ii) stacking two SSMs with multiplicative interaction, in order to compare information from previous time steps (output of the lower shifted SSMs) with the input at the current time steps – thus measuring similarity between tokens, which mimics local multiplicative interactions in linear attention. Noticeably, the combination of shift SSM and the following multiplicative interaction acts as a gate. Empirically, these solutions allow to completely solve the Induction Heads task and almost completely fit the Associative Recall, while other SSMs (e.g., S4D (Gu et al., 2022), GSS (Mehta et al., 2022)) struggle with it.

These findings show that recurrent architectures working in the state-space still have room for improvement in terms of representational capability when compared to attention-based methods. Merrill et al. (2024) showed that, similarly to Transformers, RNNs with a diagonal transition matrix (thus, most of deep SSMs) could only represent functions in TC^0. They also showed that the expressive power of SSMs can be extended by making the state transition matrices input-dependent, enabling the theoretical capability to solve NC^1-hard problems and achieving better empirical results, as evidenced by models like Liquid-S4 (Hasani et al., 2022b). Grazzi et al. (2025) demonstrate that typical linear RNNs like Mamba and DeltaNet are limited by positive-only eigenvalues in their state-transition matrices, which prevents them from solving simple tasks such as parity. By allowing negative eigenvalues and involving non-diagonal state transition matrices, they significantly improve performance on state-tracking, while maintaining stability and efficiency at scale. Concurrently, RWKV-7 (Peng et al., 2025) demonstrates remarkable expressivity, with the ability to recognize all regular languages using a small constant number of layers. This is grounded in the architecture's generalized delta rule, which incorporates vector-valued gating and in-context learning rates. These advancements build upon and extend the capabilities introduced in Gated DeltaNet (Yang et al., 2025), a linear recurrent neural network architecture that combines the strengths of Mamba2's gating mechanism for adaptive memory decay and DeltaNet's delta update rule for precise key-value memory updates. Building on Grazzi et al. (2025), Siems et al. (2025) propose DeltaProduct, a generalization of DeltaNet where each token update applies a product of multiple Householder transformations. This increases the rank of the state-transition matrix, offering a tunable balance between expressivity and efficiency.

Overcoming limitations with hybrid models The expressivity gap emphasized by Jelassi et al. (2024) highlights why SSM-based architectures still lag behind attention-only models like Mistral 7B (Jiang et al., 2023) in demanding tasks such as language modeling. To address this, recent models like Griffin (De et al., 2024) incorporate hybrid strategies, blending SSMs with local (sliding window) attention. This form of attention enables each token to access a limited context of past tokens, reducing FLOPs and constraining KV-cache sizes. In practice, local attention allows each position to attend only to a fixed number of tokens in the past, reducing the number of FLOPs with respect to standard attention and bounding the size of the KV-cache to the size of window, making it

no longer quadratic in the sequence length. Griffin matches the learning speed of Transformers (in terms of necessary training steps/samples to achieve a certain performance), and is capable to extrapolate to evaluation sequences several orders of magnitude longer than the training sequence lengths. Other architectures have adopted different hybrid schemes, layering attention and SSMs in alternation. For instance, Jamba and Jamba 1.5 (Team et al., 2024) propose a Transformer-Mamba mix, combined with a mixture-of-experts module, to achieve a balance between performance and computational cost. Poli et al. (2024) conducted a comprehensive analysis and found that a compute-optimal configuration includes roughly one-fourth self-attention layers, with the remainder composed of SSMs. Zamba (Glorioso et al., 2024), a 7B hybrid model, demonstrates strong performance relative to other open-weight models by pairing a Mamba backbone with a shared attention layer, thereby capturing the benefits of attention while keeping parameter usage minimal.

Moreover, it is of extreme interest to understand the role of model pre-training. Indeed, Amos et al. (2024) suggested that self-supervised pretraining on the task data (referred to as *self pretraining*), rather than training from scratch, significantly improves the performance on long sequences (i.e., on LRA) both in vanilla Transformers and with simple diagonal RNNs, showing the redundancy of sophisticated architectures and manually-designed biases (Gu et al., 2022; Ma et al., 2023).

8.2. Hardware-aware efficiency: FLOPs vs. memory bandwidth

Reducing the computational and memory requirements of vanilla Transformers is one of the main goal of the works described in this survey. In Section 3 we showcased several attention approximations that indeed succeed in this goal, achieving complexities linear or near-linear in sequence length (Katharopoulos et al., 2020; Peng et al., 2023; Schlag et al., 2021) by reducing the amount of computations. However, when used in practice, many of these solutions do not display substantial wall-clock speedup when compared to standard attention. Recent works such as FlashAttention (Dao, 2023; Dao et al., 2022) showed how floating-point operations (FLOPs) reduction may not correlate with fast wall-clock speed, when memory access overheads are not appropriately taken care of.

This is a direct consequence of the way the hardware of common accelerators (e.g., GPUs, TPUs, etc.) is structured. For instance, GPUs compute speed has out-paced memory speed. When considering Transformers, most operations are bottlenecked by memory accesses (Dao et al., 2022; Ivanov et al., 2021). Generally speaking, accelerators perform reads and writes to different levels of fast and slow memory, i.e., between fast GPU on-chip SRAM and relatively slow GPU high bandwidth memory (HBM). As a key example, the NVIDIA A100 is composed by 40–80 GB of HBM, with bandwidth of 1.5-2.0 TB/s and 192KB of on-chip SRAM per each of 108 streaming multiprocessors (SMs – that execute groups of thread, referred to as thread blocks, in parallel) with bandwidth estimated around 19 TB/s (Dao et al., 2022; Jia & Van Sandt, 2021). Each thread loads input from HBM to registers and on-chip SRAM for computation before writing back to HBM. Memory-bound operations (e.g., elementwise sum, reductions, softmax, batch norm, layer norm) often pose significant bottlenecks due to slower memory speeds. Vanilla Transformers store tensors such as weights and the KV-cache in HBM for the duration of the decoding. These tensors need to be transferred from HBM to the compute cores of the chip once per forward pass of the model, thus giving rise to a large amount of memory traffic.

Additionally, specialized hardware units, such as TensorCores in NVIDIA GPUs (Markidis et al., 2018) and MXUs in Google TPUs (Norrie et al., 2021), are optimized for classic architectures relying on matrix multiplications and convolutions with high FLOPs-to-byte ratios. While RNNs benefit from this due to their dense recurrence matrices, operations happening in diagonal RNNs (see SSMs and other diagonal models described in Sections 4 and 5) pose computational challenges on existing accelerators. For instance, the element-wise update equation in Griffin results in a low FLOPs-to-byte ratio, leading to memory-bound computations.

Many recent works (Gu et al., 2021b; Katsch, 2023; Sun et al., 2023a; Yang et al., 2023) propose alternative computational forms (i.e., recurrent, parallel, chunkwise) for their models, offering varying tradeoffs which can be chosen according to specific needs, but still not taking into account the I/O cost. Whilst basic implementations of the recurrent forms minimize the total FLOPs, they incur in high I/O costs due to storing 2D hidden states in HBM (Mao, 2022). Katharopoulos et al. (2020) reduce this cost by avoiding state materialization and recomputing hidden states during the backward pass (hereinafter referred to as *recomputation*), while Katsch (2023) leverages the parallel scan algorithm to parallelize a recurrent form over the sequence length. However, the absence of parallelizable operations, such as matrix multiplications, hinder the benefits from Tensorcores, thus not translating to actual wall-time efficiency. In contrast, the quadratic-complexity form of Transformers exhibits high FLOPs, making long-sequence training expensive, even if Qin et al. (2023a) showed that it can be as efficient as FlashAttention. The chunkwise form (see Hua et al. (2022), Sun et al. (2023a)) provides an interesting compromise, allowing fine-grained optimization by interpolating between the parallel and recurrent forms with the user-selected chunk size. Differently from the recurrent form, most operations can be done via matmuls, enabling the use of tensor cores. However, most implementations are not I I/O-aware and thus slower than FlashAttention for moderate sequence lengths (e.g., 2K-4K).

When considering solutions to the aforementioned issues, FlashAttention (Dao et al., 2022) showed how even vanilla Transformers could benefit from I/O aware techniques. The authors of Dao et al. (2022) restructure (*tiling*) the attention computation to prevent reading and writing (*materializing*) the large $L \times L$ attention matrix to and from the slow HBM. Meanwhile, in large-scale training and long-sequence modeling scenarios where the batch size tends to be small, parallelizing over the temporal dimension enables high GPU occupancy (Dao, 2023).

Following similar intuitions, few recent works are starting to devise novel I/O-aware mechanisms. The GLA model proposed by Yang et al. (2023) (see Section 3) borrows concepts from Linear Transformers, gating mechanism and FlashAttention, resulting in the proposal of FlashLinearAttention, an hardware-efficient implementation of Linear Attention in a chunkwise form, also generalized to a more expressive variant with data-dependent gates. The authors propose a memory-efficient version where chunk-level states are not materialized in HBM memory, by leveraging tiling and recomputation, and a materialization version, which leverages sequence-level parallelism and thus allows for higher training throughput at the cost of a slightly increasing memory footprint. Overall, FlashLinearAttention is faster than FlashAttention-2 (Dao, 2023), despite the introduction of an expressive gating mechanism. A concurrent work by Qin et al. (2024) also proposed an I/O-aware version of linear attention, which is similar to the non-materialization version of FlashLinearAttention.

The selection mechanism underlying Mamba (Gu & Dao, 2023) results in a time-varying SSM. Time varying solutions (such as Gu et al. (2020a)) had been previously overtaken by LTI (non-selective) models (S4, S5 and all derivatives) due to the fact that the latter can be efficiently expressed in the form of global convolutions or associative scans, while the former require to compute and materialize the large latent state into HBM. However, the additional expressive power achieved by means of “selection” procedures, calls for the need of an efficient hardware-aware implementation. Indeed, similarly to GLA, Mamba leverages kernel fusion (to reduce the amount of memory I/Os), parallel scan (to avoid slowdowns due to the sequential recurrence), and recomputation (the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from HBM to SRAM). Overall, Mamba shows the same memory requirements as an optimized transformer with FlashAttention.

To address the low FLOPs-to-byte ratio due to elementwise operations, the Griffin model (De et al., 2024) is implemented in TPUs and

leverages a custom kernel for the computation of the state update equation exploiting linear scans. The implemented kernel minimizes memory transfers by keeping hidden states in the TPUs Vector Memory (VMEM) instead of HBM and performing memory transfers in larger chunks, resulting in a 3× speedup with respect to a vanilla implementation. As an interesting remark, the gating mechanism in Griffin is not compatible with the usual convolutional view or associative scans proper of LTI SSMs. Even if the authors could have used associative scans similarly to GateLoop (Katsch, 2023), they found that these operations reduce the number of FLOPs, but do not reduce memory overheads, which is their primary bottleneck. Summing up, when dealing with long-sequence processing, a very relevant role is played by the way computations are optimized, and identifying primary performance bottlenecks on the target hardware or modern accelerators represents a fundamental step for future research. Novel models in this research field could greatly benefit from ad-hoc hardware and algorithmic solutions.

8.3. Infinite-length sequences: lifelong learning

This survey describes multiple architectural and algorithmic solutions that allow neural networks to handle sequences of increasing length, by reducing the overall computational complexities and leveraging a stateful recurrent representation. In the previous subsections, we showcased two very relevant research avenues for future studies, i.e., the expressivity improvement of recurrence-based models with respect to full-fledged attention, and the need for hardware-aware solutions capable of effectively dealing with very long sequences. However, there is still a significant gap that requires specific studies to move beyond current technologies, that shows up when thinking of transitioning from *long* sequences to *infinite-length* sequences. Indeed, while researchers aim to mimic human learning abilities from sequential data, the most common existing approaches primarily deal with finite-length datasets BPTT in stateful models, or standard Back-Propagation, whilst the alternative learning algorithms we described in Section 6 are still not widespread in the literature. This stands in stark contrast to human learning from continuous perceptual stimuli, which unfold over time without being pre-segmented into finite-length sequences for stochastic gradient descent optimization, and without the need to explicitly store exemplars into buffers or the need of building data collections (Gori & Melacci, 2023). Moreover, this misalignment deviates from the principles of seminal works in recurrent models. Indeed, the goal of works such as the one by Williams and Peng (1990) was to devise *"an online algorithm, designed to be used to train a network while it runs; no manual state resets or segmentations of the training stream are required"*. Even the original formulations of LSTMs, among others (Hochreiter & Schmidhuber, 1997), were introduced with a learning algorithm that unlike BPTT is *"local in space and time"*, where *"there is no need to store activation values observed during sequence processing in a stack with potentially unlimited size"*.

In contrast, as we described in the previous Sections, the most popular research activities in the field of RNNs are not considering the case of learning *online* from a continuous, possibly infinite, stream of data (i.e., infinite-length sequence), without resets or segmentations (Betti et al., 2022). This extremely challenging scenario involves two primary demands: (*i*.) computing meaningful gradients at time t based on processed data up to that point, without fully unrolling the network across the entire (potentially infinite) input sequence (in some cases, only relying on information from times t and $t-1$), and updating model parameters at each time step; (*ii*.) possessing the ability to adapt to the characteristics of the streamed data, retaining acquired skills over time, as emphasized in continual learning contexts (Parisi & Lomonaco, 2020). In Section 6 we described the scientific efforts aimed at developing alternatives to BPTT. We showcased online learning algorithms for RNNs, particularly addressing requirement (*i*.). RTRL stands as the most recognized method for online learning with recurrent models, which has been recently refined to mitigate computational and memory constraints (Irie et al., 2023b), alongside various direct or indirect adaptations (Marschall et al., 2020). We described recent studies that explore forward computation of gradients, enabled by state-space modeling (Zucchet et al., 2023b), albeit reliant on finite-length sequences processed without updating learnable parameters at each time step, thereby not fully satisfying requirement (*i*.). Considering demand (*ii*.) leads to a scenario akin to Online Continual Learning (Parisi & Lomonaco, 2020) with substantial overlap with Streaming Learning, as outlined in Gunasekara et al. (2023). It's important to distinguish this from approaches focused solely on continual learning from streams of finite-length sequences (Cossu et al., 2021; Ehret et al., 2020).

Summing up, the scenario described in this section remains largely unexplored. Despite advancements in related domains, meeting requirements (*i*.) and (*ii*.) independently poses significant challenges, and their combined fulfillment presents an even more daunting task. Nonetheless, this scenario reflects the reality of learning from continuous streams of visual data, auditory stimuli, perpetual video feeds, language inputs, and more. Whilst common benchmarks for sequential data (see Section 7) do provide valuable opportunities to assess the fulfillment of requirement (*i*.), it is crucial to operate within the typical framework of online learning problems, where all data contributes to training and errors accumulate during learning.[33] Certain tasks such as time-series forecasting offer greater flexibility in this context, compared to classification tasks, due to the fact that they provide ground truth at each t. Additionally, the properties of the series might change over time, partially allowing exploration of requirement (*ii*.). However, a comprehensive evaluation of property (*ii*.) remains challenging with existing datasets.

All these considerations suggest that there is room for future research. Providing sources for streaming data becomes extremely important, in a context where data privacy and storage become more and more relevant (Gori & Melacci, 2023). For instance, in computer vision, a paradigm-shift from image-based processing to videos is happening (Betti et al., 2020; Brooks et al., 2024; Carreira et al., 2023; Faggi et al., 2023; Meloni et al., 2021; Selva et al., 2023; Tiezzi et al., 2022), thus requiring to rethink data handling pipelines and data engineering. From the pure machine learning side, research on novel models and learning methods which are better at capturing regularities on the input stream in an online manner are needed. At the same time, model evaluation and selection procedures must evolve (Ghunaim et al., 2023), to be capable of handling settings where models self-adapt their parameters over time continuously, potentially in a truly lifelong learning manner.

9. Conclusions

While Transformers initially overshadowed Recurrent Neural Networks, recent developments in large-context Transformers and (deep) State-Space Models have highlighted the importance of recurrent computations as a viable road to go beyond the limitations of Transformers both in terms of performance on long sequences and scalability. This survey outlines the latest trends and approaches for sequential data processing, offering insights into recent architectural and novel algorithmic solutions.

Interestingly, the recent literature in the context of recurrent long-context Transformers and State-Space Models share the same features, leading towards a computational model based on stateful computations, confirming the advantages brought by the idea of working in the state space. We surveyed the most important works, and we discussed novel trends in learning algorithms for sequential data, offering a broad and detailed perspective on the last generation of neural architectures. This survey also outlined novel research opportunities. In fact, we emphasized how the research direction of lifelong online learning from a stream of data, intended to be an infinite-length sequence, represents

[33] The input at time u_t is processed and the produced output y_t is compared with the available ground truth to update the accuracy estimate; then, y_t is used to evaluate a loss function, compute gradients at t and update the model.

a challenge that is still very open and might be faced starting from this renewed interest in recurrent models.

CRediT authorship contribution statement

Matteo Tiezzi: Writing – review & editing, Writing – original draft, Visualization, Methodology, Investigation, Formal analysis, Conceptualization; **Michele Casoni:** Writing – review & editing, Visualization, Investigation; **Alessandro Betti:** Writing – review & editing; **Marco Gori:** Supervision; **Stefano Melacci:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partly supported by the University of Siena (Piano per lo Sviluppo della Ricerca - PSR 2024, F-NEW FRONTIERS 2024), under the project “TIme-driveN StatEful Lifelong Learning” (TINSELL) and also by the project “CONSTR: a COllectionless-based Neuro-Symbolic Theory for learning and Reasoning”, PARTENARIATO ESTESO “Future Artificial Intelligence Research - FAIR”, SPOKE 1 “Human-Centered AI” Università di Pisa, “NextGenerationEU”, CUP I53C22001380006.

References

Almeida, L. B. (1990). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning* (pp. 102–111).

Alonso, C. A., Sieber, J., & Zeilinger, M. N. (2024). State space models as foundation models: A control theoretic overview. arXiv:2403.16899.

Amos, I., Berant, J., & Gupta, A. (2024). Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. In *The twelfth international conference on learning representations*. https://openreview.net/forum?id=PdaPky8MUn.

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2012). Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Ambient assisted living and home care: 4th international workshop, IWAAL 2012, Vitoria-Gasteiz, Spain, December 3–5, 2012. proceedings 4* (pp. 216–223). Springer.

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International conference on machine learning* (pp. 1120–1128). PMLR.

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., & Ré, C. (2023). Zoology: Measuring and improving recall in efficient language models. arXiv:2312.04927.

Askari, A., Negiar, G., Sambharya, R., & Ghaoui, L. E. (2018). Lifted neural networks. arXiv:1805.01532. https://arxiv.org/abs/1805.01532.

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., Assael, Y., Jumper, J., Kohli, P., & Kelley, D. R. (2021). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, *18*(10), 1196–1203.

Azencot, O., Erichson, N. B., Lin, V., & Mahoney, M. (2020). Forecasting sequential data using consistent koopman autoencoders. In I. H. Daumé, & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 475–485). PMLR (*vol. 119*). Proceedings of Machine Learning Research.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., & Ionescu, C. (2016a). Using fast weights to attend to the recent past. *Advances in Neural Information Processing Systems*, *29*.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016b). Layer normalization. arXiv:1607.06450.

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., & Baskurt, A. (2011). Sequential deep learning for human action recognition. In *Human behavior understanding: Second international workshop, HBU 2011, Amsterdam, the Netherlands, November 16, 2011. proceedings 2* (pp. 29–39). Springer.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv:1409.0473.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271.

Balduzzi, D., & Ghifary, M. (2016). Strongly-typed recurrent neural networks. In *International conference on machine learning* (pp. 1292–1300). PMLR.

Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., & Hochreiter, S. (2024). xLSTM: Extended long short-term memory. arXiv:2405.04517.

Becker, P., Pandya, H., Gebhardt, G., Zhao, C., Taylor, C. J., & Neumann, G. (2019). Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In *International conference on machine learning* (pp. 544–552). PMLR.

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv:2004.05150.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Betti, A., Faggi, L., Gori, M., Tiezzi, M., Marullo, S., Meloni, E., & Melacci, S. (2022). Continual learning through hamilton equations. In *Conference on lifelong learning agents* (pp. 201–212). PMLR (*vol. 199*).

Betti, A., Gori, M., & Melacci, S. (2020). Learning visual features under motion invariance. *Neural Networks*, *126*, 275–299.

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). Recurrent neural networks for short-term load forecasting - An overview and comparative analysis. Springer Briefs in Computer Science. Springer https://doi.org/10.1007/978-3-319-70338-1

Bisk, Y., Zellers, R., Gao, J., Choi, Y. et al. (2020). Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence* (pp. 7432–7439). (*vol. 34*).

Blelloch, G. E. (1990). Prefix sums and their applications.

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H. et al. (2014). Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation* (pp. 12–58).

Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2016). Quasi-recurrent neural networks. arXiv:1611.01576.

Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., Ng, C., Wang, R., & Ramesh, A. (2024). Video generation models as world simulators. https://openai.com/research/video-generation-models-as-world-simulators.

Brown, A. E. X., Yemini, E. I., Grundy, L. J., Jucikas, T., & Schafer, W. R. (2013). A dictionary of behavioral motifs reveals clusters of genes affecting caenorhabditis elegans locomotion. *Proceedings of the National Academy of Sciences*, *110*(2), 791–796.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.

Bulatov, A., Kuratov, Y., & Burtsev, M. (2022). Recurrent memory transformer. *Advances in Neural Information Processing Systems*, *35*, 11079–11091.

Bulatov, A., Kuratov, Y., & Burtsev, M. S. (2023). Scaling transformer to 1m tokens and beyond with RMT. arXiv:2304.11062.

Carreira, J., King, M., Pătrăucean, V., Gokay, D., Ionescu, C., Yang, Y., Zoran, D., Heyward, J., Doersch, C., Aytar, Y. et al. (2023). Learning from one continuous video stream. arXiv:2312.00598.

Carreira-Perpinan, M., & Wang, W. (2014). Distributed optimization of deeply nested systems. In *Artificial intelligence and statistics* (pp. 10–19). PMLR.

Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., & Federico, M. (2014). Report on the 11th IWSLT evaluation campaign. In *Proceedings of the 11th international workshop on spoken language translation: Evaluation campaign* (pp. 2–17).

Chang, B., Chen, M., Haber, E., & Chi, E. H. (2018). AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International conference on learning representations*.

Chen, K., Wang, R., Utiyama, M., & Sumita, E. (2019). Recurrent positional embedding for neural machine translation. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 1361–1367).

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, *34*, 15084–15097.

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Schuster, M., Shazeer, N., Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Chen, Z., Wu, Y., Hughes, M., (2018a). The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 76–86).

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018b). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, *31*.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., & Bengio, Y. (2018). Babyai: A platform to study the sample efficiency of grounded language learning. arXiv:1810.08272.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Davis, J., Sarlos, T., Belanger, D., Colwell, L., & Weller, A. (2020a). Masked language modeling for proteins via linearly scalable long-context transformers. arXiv:2006.03555.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D.B., Colwell, L. J., Weller, A. (2020b). Rethinking attention with performers. In *International conference on learning representations*.

Chung, S., & Siegelmann, H. (2021). Turing completeness of bounded-precision recurrent neural networks. *Advances in Neural Information Processing Systems*, *34*, 28431–28441.

Ciarlet, P. G. (2013). Linear and nonlinear functional analysis with applications (vol. 130). Siam.

Cirone, N. M., Orvieto, A., Walker, B., Salvi, C., & Lyons, T. (2024). Theoretical foundations of deep selective state-space models. arXiv:2402.19047.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., & Toutanova, K. (2019). Boolq: Exploring the surprising difficulty of natural yes/no questions. arXiv:1905.10044.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., & Tafjord, O. (2018). Think you have solved question answering? Try arc, the ai2 reasoning challenge. arXiv:1803.05457.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv:1511.07289.

Conway, C. M., & Christiansen, M. H. (2001). Sequential learning in non-human primates. *Trends in Cognitive Sciences*, *5*(12), 539–546.

Cossu, A., Carta, A., Lomonaco, V., & Bacciu, D. (2021). Continual learning for recurrent neural networks: An empirical evaluation. *Neural Networks*, *143*, 607–627.

Crick, F. (1989). The recent excitement about neural networks. *Nature*, *337*(6203), 129–132.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 2978–2988).

Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv:2307.08691.

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, *35*, 16344–16359.

Dao, T., Fu, D. Y., Saab, K. K., Thomas, A. W., Rudra, A., & Ré, C. (2023). Hungry hungry hippos: Towards language modeling with state space models. In *Proceedings of the 11th international conference on learning representations (ICLR)*.

Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*. PMLR.

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S. et al. (2024). Griffin: Mixing gated linear recurrences with local attention for efficient language models. arXiv:2402.19427.

Didolkar, A., Gupta, K., Goyal, A., Gundavarapu, N. B., Lamb, A. M., Ke, N. R., & Bengio, Y. (2022). Temporal latent bottleneck: Synthesis of fast and slow processing mechanisms in sequence learning. *Advances in Neural Information Processing Systems*, *35*, 10505–10520.

Ding, S., Shang, J., Wang, S., Sun, Y., Tian, H., Wu, H., & Wang, H. (2021). Ernie-doc: A retrospective long-document modeling transformer. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: Long papers)* (pp. 2914–2927).

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Houlsby, N., Gelly, S., Zhang, X., & Uszkoreit, J. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv:2010.11929.

Dufter, P., Schmitt, M., & Schütze, H. (2022). Position information in transformers: An overview. *Computational Linguistics*, *48*(3), 733–763. https://aclanthology.org/2022.cl-3.7. https://doi.org/10.1162/coli_a_00445

Ehret, B., Henning, C., Cervera, M., Meulemans, A., Von Oswald, J., & Grewe, B. F. (2020). Continual learning in recurrent neural networks. In *International conference on learning representations*.

Eisenmann, L., Monfared, Z., Göring, N., & Durstewitz, D. (2023). Bifurcations and loss jumps in RNN training. *Advances in Neural Information Processing Systems*, *36*, 70511–70547.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K. et al. (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*, *1*, 1.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Elman, J. L., & Zipser, D. (1988). Learning the hidden structure of speech. *The Journal of the Acoustical Society of America*, *83*(4), 1615–1626.

Erichson, N. B., Azencot, O., Queiruga, A. F., Hodgkinson, L., & Mahoney, M. W. (2021). Lipschitz recurrent neural networks. In *9th international conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021*. OpenReview.net. https://openreview.net/forum?id=-N7PBXqOUJZ.

Faggi, L., Betti, A., Zanca, D., Melacci, S., & Gori, M. (2023). Local propagation of visual stimuli in focus of attention. *Neurocomputing*, *560*, 126775.

Frasconi, P., Gori, M., & Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*(5), 768–786. https://doi.org/10.1109/72.712151

Gallicchio, C., & Micheli, A. (2011). Architectural and markovian factors of echo state networks. *Neural Networks*, *24*(5), 440–456. https://doi.org/10.1016/j.neunet.2011.02.002

Gallicchio, C., & Micheli, A. (2017). Deep echo state network (deepesn): A brief survey. arXiv:1712.04323.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N. et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. arXiv:2101.00027.

Gao, L., Tow, S., Black, S., DiPofi, A., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, L., Reynolds, L., Tang, E., Thite, A., Wang, B., Zou, A., A framework for few-shot language model evaluation (2021). https://doi.org/10.5281/zenodo.5371629

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., & Pallett, D. S. (1993). Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon Technical Report n*, *93*, 27403.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International conference on machine learning* (pp. 1243–1252). PMLR.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, *12*(10), 2451–2471.

Ghunaim, Y., Bibi, A., Alhamoud, K., Alfarra, M., Al Kader Hammoud, H. A., Prabhu, A., Torr, P. H. S., & Ghanem, B. (2023). Real-time evaluation in online continual learning: A new hope. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11888–11897).

Glorioso, P., Anthony, Q., Tokpanov, Y., Whittington, J., Pilault, J., Ibrahim, A., & Millidge, B. (2024). Zamba: A compact 7b SSM hybrid model. arXiv:2405.16712.

Goel, K., Gu, A., Donahue, C., & Ré, C. (2022). It's raw! Audio generation with state-space models. In *International conference on machine learning* (pp. 7616–7633). PMLR.

Gori, M., & Melacci, S. (2023). Collectionless artificial intelligence. arXiv: arXiv 2309.06938.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.* (pp. 729–734). IEEE (*vol. 2*).

Graves, A., & Graves, A. (2012). Supervised sequence labelling. Springer.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. arXiv:1410.5401.

Grazzi, R., Siems, J., Zela, A., Franke, J. K. H., Hutter, F., & Pontil, M. (2025). Unlocking state-tracking in linear RNNs through negative eigenvalues. In *The thirteenth international conference on learning representations*.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., & Graves, A. (2016). Memory-efficient backpropagation through time. *Advances in Neural Information Processing Systems*, *29*.

Gu, A., & Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. arXiv:2312.00752.

Gu, A., Dao, T., Ermon, S., Rudra, A., & Ré, C. (2020a). Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, *33*, 1474–1487.

Gu, A., Goel, K., Gupta, A., & Ré, C. (2022). On the parameterization and initialization of diagonal state space models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 35971–35983). Curran Associates, Inc. (*vol. 35*). https://proceedings.neurips.cc/paper_files/paper/2022/file/e9a32fade47b906de908431991440f7c-Paper-Conference.pdf.

Gu, A., Goel, K., & Re, C. (2021a). Efficiently modeling long sequences with structured state spaces. In *International conference on learning representations*.

Gu, A., Gulcehre, C., Paine, T., Hoffman, M., & Pascanu, R. (2020b). Improving the gating mechanism of recurrent neural networks. In *International conference on machine learning* (pp. 3800–3809). PMLR.

Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., & Ré, C. (2021b). Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in Neural Information Processing Systems*, *34*, 572–585.

Gu, C., Sun, C., Ross, D. A., Vondrick, C., Pantofaru, C., Li, Y., Vijayanarasimhan, S., Toderici, G., Ricco, S., Sukthankar, R. et al. (2018). Ava: A video dataset of spatio-temporally localized atomic visual actions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 6047–6056).

Gu, F., Askari, A., & Ghaoui, L. E. (2020c). Fenchel lifted networks: A lagrange relaxation of neural network training. In S. Chiappa, & R. Calandra (Eds.), *Proceedings of the twenty third international conference on artificial intelligence and statistics* (pp. 3362–3371). PMLR (*vol. 108*). Proceedings of Machine Learning Research. https://proceedings.mlr.press/v108/gu20a.html.

Gunasekara, N., Pfahringer, B., Gomes, H. M., & Bifet, A. (2023). Survey on online streaming continual learning. In *Proceedings of the thirty-second international joint conference on artificial intelligence* (pp. 6628–6637).

Gupta, A., Gu, A., & Berant, J. (2022a). Diagonal state spaces are as effective as structured state spaces. In *Advances in neural information processing systems*.

Gupta, A., Mehta, H., & Berant, J. (2022b). Simplifying and understanding state space models with diagonal linear RNNs. arXiv:2212.00768.

Hahn, M. (2020). Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, *8*, 156–171.

Hanson, S. J., & Kegl, J. (1987). Parsnip: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the eight annual meeting of the cognitive science society* (pp. 106–119). Lawrence Erlbaum Associates Hillsdale, NJ.

Hao, J., Wang, X., Yang, B., Wang, L., Zhang, J., & Tu, Z. (2019). Modeling recurrence for transformer. In *Proceedings of NAACL-HLT* (pp. 1198–1207).

Hao, Y., Angluin, D., & Frank, R. (2022). Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, *10*, 800–810.

Hasani, R., Lechner, M., Amini, A., Liebenwein, L., Ray, A., Tschaikowski, M., Teschl, G., & Rus, D. (2022a). Closed-form continuous-time neural networks. *Nature Machine Intelligence*, *4*(11), 992–1003.

Hasani, R., Lechner, M., Amini, A., Rus, D., & Grosu, R. (2021). Liquid time-constant networks. In *Proceedings of the AAAI conference on artificial intelligence* (pp. 7657–7666). (*vol. 35*).

Hasani, R., Lechner, M., Wang, T.-H., Chahine, M., Amini, A., & Rus, D. (2022b). Liquid structural state-space models. arXiv:2209.12951.

Hè, H., & Kabic, M. (2023). A unified view of long-sequence models towards million-scale dependencies. arXiv:2302.06218.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2020). Measuring massive multitask language understanding. arXiv:2009.03300.

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv:1606.08415.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, *91*(1), 31.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A. et al. (2022). An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, *35*, 30016–30030.

Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, *36*(3), 1171–1220. https://doi.org/10.1214/009053607000000677

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W. et al. (2021). LoRA: Low-rank adaptation of large language models. In *International conference on learning representations*.

Hua, W., Dai, Z., Liu, H., & Le, Q. (2022). Transformer quality in linear time. In *International conference on machine learning* (pp. 9099–9117). PMLR.

Huang, F., Lu, K., Yuxi, C., Qin, Z., Fang, Y., Tian, G., & Li, G. (2022). Encoding recurrence into transformers. In *The eleventh international conference on learning representations*.

Huang, Y., Bai, Y., Zhu, Z., Zhang, J., Zhang, J., Su, T., Liu, J., Lv, C., Zhang, Y., Fu, Y. et al. (2024). C-eval: A multi-level multi-discipline Chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, *36*.

Hunter, J. S. (1986). The exponentially weighted moving average. *Journal of Quality Technology*, *18*(4), 203–210.

Hutchins, D., Schlag, I., Wu, Y., Dyer, E., & Neyshabur, B. (2022). Block-recurrent transformers. *Advances in Neural Information Processing Systems*, *35*, 33248–33261.

Irie, K., Csordás, R., & Schmidhuber, J. (2023a). Practical computational power of linear transformers and their recurrent and self-referential extensions. arXiv:2310.16076.

Irie, K., Faccio, F., & Schmidhuber, J. (2022a). Neural differential equations for learning to program neural nets through continuous learning rules. *Advances in Neural Information Processing Systems*, *35*, 38614–38628.

Irie, K., Gopalakrishnan, A., & Schmidhuber, J. (2023b). Exploring the promise and limits of real-time recurrent learning. arXiv:2305.19044.

Irie, K., Schlag, I., Csordás, R., & Schmidhuber, J. (2021). Going beyond linear transformers with recurrent fast weight programmers. *Advances in Neural Information Processing Systems*, *34*, 7703–7717.

Irie, K., Schlag, I., Csordás, R., & Schmidhuber, J. (2022b). A modern self-referential weight matrix that learns to modify itself. In *International conference on machine learning* (pp. 9660–9677). PMLR.

Ivanov, A., Dryden, N., Ben-Nun, T., Li, S., & Hoefler, T. (2021). Data movement is all you need: A case study on optimizing transformers. *Proceedings of Machine Learning and Systems*, *3*, 711–732.

Javed, K., Shah, H., Sutton, R., & White, M. (2021a). Scalable online state construction using recurrent networks, Blogpost.

Javed, K., Shah, H., Sutton, R., & White, M. (2023). Online real-time recurrent learning using sparse connections and selective learning. arXiv:2302.05326.

Javed, K., White, M., & Sutton, R. (2021b). Scalable online recurrent learning using columnar neural networks. arXiv:2103.05787.

Jelassi, S., Brandfonbrener, D., Kakade, S. M., & Malach, E. (2024). Repeat after me: Transformers are better than state space models at copying. In *Forty-first international conference on machine learning*.

Jia, Z., & Van Sandt, P. (2021). Dissecting the ampere GPU architecture via microbenchmarking. In *GPU technology conference*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las, C. D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L. et al. (2023). Mistral 7b. arXiv:2310.06825.

Jing, L., Gulcehre, C., Peurifoy, J., Shen, Y., Tegmark, M., Soljacic, M., & Bengio, Y. (2019). Gated orthogonal recurrent units: On learning to forget. *Neural Computation*, *31*(4), 765–783.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In *Advances in psychology* (pp. 471–495). Elsevier (*vol. 121*).

Jurtz, V. I., Johansen, A. R., Nielsen, M., Almagro Armenteros, J. J., Nielsen, H., Sønderby, C. K., Winther, O., & Sønderby, S. K. (2017). An introduction to deep learning on biological sequence data: Examples and solutions. *Bioinformatics (Oxford, England)*, *33*(22), 3685–3690.

Kag, A., & Saligrama, V. (2021a). Time adaptive recurrent neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 15149–15158).

Kag, A., & Saligrama, V. (2021b). Training recurrent neural networks via forward propagation through time. In *International conference on machine learning* (pp. 5189–5200). PMLR.

Kag, A., Zhang, Z., & Saligrama, V. (2019). Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International conference on learning representations*.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. arXiv:2001.08361.

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., & Smith, N. A. (2021). Finetuning pretrained transformers into RNNs. In *Proceedings of the 2021 conference on empirical methods in natural language processing* (pp. 10630–10643).

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning* (pp. 5156–5165). PMLR.

Katsch, T. (2023). Gateloop: Fully data-controlled linear recurrence for sequence modeling. arXiv:2311.01927.

Keles, F. D., Wijewardena, P. M., & Hegde, C. (2023). On the computational complexity of self-attention. In *International conference on algorithmic learning theory* (pp. 597–619). PMLR.

Keller, T. A., Muller, L., Sejnowski, T., & Welling, M. (2023). Traveling waves encode the recent past and enhance sequence learning. arXiv:2309.08045.

Keller, T. A., & Welling, M. (2023). Neural wave machines: Learning spatiotemporally structured representations with locally coupled oscillatory recurrent neural networks.

Kenton, J. D. M.-W. C., & Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT* (p. 2). (*vol. 1*).

Kerg, G., Goyette, K., Puelma Touzel, M., Gidel, G., Vorontsov, E., Bengio, Y., & Lajoie, G. (2019). Non-normal recurrent neural network (nnrnn): Learning long time dependencies while improving expressivity with transient dynamics. *Advances in Neural Information Processing Systems*, *32*.

Kidger, P. (2022). On neural differential equations. arXiv:2202.02435.

Kidger, P., Morrill, J., Foster, J., & Lyons, T. (2020). Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, *33*, 6696–6707.

Kim*, J., Linsley*, D., Thakkar, K., & Serre, T. (2020). Disentangling neural mechanisms for perceptual grouping. In *International conference on learning representations*.

Kitaev, N., Kaiser, L., & Levskaya, A. (2020). Reformer: The efficient transformer. In *International conference on learning representations*. https://openreview.net/forum?id=rkgNKkHtvB.

Kitagawa, G. (1998). A self-organizing state-space model. *Journal of the American Statistical Association*, (pp. 1203–1215).

Kocmi, T., Bawden, R., Bojar, O., Dvorkovich, A., Federmann, C., Fishel, M., Gowda, T., Graham, Y., Grundkiewicz, R., Haddow, B. et al. (2022). Findings of the 2022 conference on machine translation (WMT22). In *Proceedings of the seventh conference on machine translation (WMT)* (pp. 1–45).

Koopman, B. O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, *17*(5), 315–318.

Krizhevsky, A., Hinton, G. et al. (2009). Learning multiple layers of features from tiny images. *University of Toronto Technical Report*,

Kuriscak, E., Marsalek, P., Stroffek, J., & Toth, P. G. (2015). Biological context of hebb learning in artificial neural networks, a review. *Neurocomputing*, *152*, 27–35.

Lanthaler, S., Rusch, T. K., & Mishra, S. (2023). Neural oscillators are universal. arXiv:2305.08753.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. arXiv:1504.00941.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lei, J., Wang, L., Shen, Y., Yu, D., Berg, T., & Bansal, M. (2020). Mart: Memory-augmented recurrent transformer for coherent video paragraph captioning. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 2603–2614).

Lei, T., Zhang, Y., Wang, S. I., Dai, H., & Artzi, Y. (2018). Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 conference on empirical methods in natural language processing* (pp. 4470–4481).

Lezcano-Casado, M., & Martınez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International conference on machine learning*. PMLR.

Li, H., Zhang, Y., Koto, F., Yang, Y., Zhao, H., Gong, Y., Duan, N., & Baldwin, T. (2023a). Cmmlu: Measuring massive multitask language understanding in chinese. arXiv:2306.09212.

Li, J., Tang, T., Zhao, W. X., Nie, J.-Y., & Wen, J.-R. (2022a). Pretrained language models for text generation: A survey. arXiv:2201.05273.

Li, S., Li, W., Cook, C., Zhu, C., & Gao, Y. (2018). Independently recurrent neural network (indrnn): building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5457–5466).

Li, X., Li, Z., Luo, X., Xie, H., Lee, X., Zhao, Y., Wang, F. L., & Li, Q. (2023b). Recurrent attention networks for long-text modeling. In A. Rogers, J. L. Boyd-Graber, & N. Okazaki (Eds.), *Findings of the association for computational linguistics: ACL 2023, Toronto, Canada, July 9–14, 2023* (pp. 3006–3019). Association for Computational Linguistics https://doi.org/10.18653/V1/2023.FINDINGS-ACL.188

Li, Y., Cai, T., Zhang, Y., Chen, D., & Dey, D. (2022b). What makes convolutional models great on long sequence modeling? arXiv:2210.09298.

Li, Z., Han, J., E, W., & Li, Q. (2022c). Approximation and optimization theory for linear continuous-time recurrent neural networks. *The Journal of Machine Learning Research*, *23*(1), 1997–2081.

Li, Z., Han, J., Li, Q. et al. (2020). On the curse of memory in recurrent neural networks: approximation and optimization analysis. arXiv:2009.07799.

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., & Zemel, R. (2018). Reviving and improving recurrent back-propagation. In *International conference on machine learning* (pp. 3082–3091). PMLR.

Lillicrap, T. P., & Santoro, A. (2019). Backpropagation through time and the brain. *Current opinion in neurobiology*, *55*, 82–89.

Lim, S. H., Erichson, N. B., Hodgkinson, L., & Mahoney, M. W. (2021). Noisy recurrent neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 5124–5137). Curran Associates, Inc. (*vol. 34*). https://proceedings.neurips.cc/paper_files/paper/2021/file/29301521774ff3cbd26652b2d5c95996-Paper.pdf.

Lim, Y. H., Zhu, Q., Selfridge, J., & Kasim, M. F. (2023). Parallelizing non-linear sequential models over the sequence length. arXiv:2309.12252.

Lin, T., Wang, Y., Liu, X., & Qiu, X. (2022). A survey of transformers. *AI Open*, *3*, 111–132. https://doi.org/10.1016/j.aiopen.2022.10.001

Linsley, D., Karkada Ashok, A., Govindarajan, L. N., Liu, R., & Serre, T. (2020). Stable and expressive recurrent vision models. *Advances in Neural Information Processing Systems*, *33*, 10456–10467.

Linsley, D., Kim, J., Veerabadran, V., Windolf, C., & Serre, T. (2018). Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in Neural Information Processing Systems*, *31*.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. arXiv:1506.00019.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., & Zhang, C. (2022a). Transformers learn shortcuts to automata. arXiv:2210.10749.

Liu, J., Pan, Z., He, H., Cai, J., & Zhuang, B. (2022b). Ecoformer: Energy-saving attention with linear complexity. *Advances in Neural Information Processing Systems*, *35*, 10295–10308.

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., & Zettlemoyer, L. (2023). Mega: Moving average equipped gated attention. In *The eleventh international conference on learning representations, ICLR 2023, Kigali, Rwanda, May 1–5, 2023*. OpenReview.net. https://openreview.net/pdf?id=qNLe3iq2El.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 142–150). Portland, Oregon, USA: Association for Computational Linguistics. http://www.aclweb.org/anthology/P11-1015.

MacKay, M., Vicol, P., Ba, J., & Grosse, R. B. (2018). Reversible recurrent neural networks. *Advances in Neural Information Processing Systems*, *31*.

Mackey, M. C., & Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science (New York, N.Y.)*, *197*(4300), 287–289.

Maggini, M., Tiezzi, M., & Gori, M. (2024). A lagrangian framework for learning in graph neural networks. In *Artificial intelligence in the age of neural networks and brain computing* (pp. 343–365). Elsevier.

Mahoney, M. (2011). Large text compression benchmark.

Mai, Z., Li, R., Jeong, J., Quispe, D., Kim, H., & Sanner, S. (2022). Online continual learning in image classification: An empirical survey. *Neurocomputing*, *469*, 28–51.

Mandic, D. P., & Chambers, J. A. (2001). Recurrent neural networks for prediction: Learning algorithms, architectures and stability (vol. 130). John Wiley & Sons.

Mao, H. H. (2022). Fine-tuning pre-trained transformers into decaying fast weights. In *Proceedings of the 2022 conference on empirical methods in natural language processing* (pp. 10236–10242).

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The penn treebank .

Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B., & Vetter, J. S. (2018). Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (pp. 522–531). IEEE.

Marra, G., Tiezzi, M., Melacci, S., Betti, A., Maggini, M., & Gori, M. (2020). Local propagation in constraint-based neural networks. In *2020 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Marschall, O., Cho, K., & Savin, C. (2020). A unified framework of online learning algorithms for training recurrent neural networks. *The Journal of Machine Learning Research*, *21*(1), 5320–5353.

Martin, E., & Cundy, C. (2018). Parallelizing linear recurrent neural nets over sequence length. In *International conference on learning representations*.

Massaroli, S., Poli, M., Fu, D. Y., Kumbong, H., Parnichkun, R. N., Timalsina, A., Romero, D. W., McIntyre, Q., Chen, B., Rudra, A. et al. (2023). Laughing hyena distillery: Extracting compact recurrences from convolutions. arXiv:2310.18780.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*, 115–133.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., & Bengio, Y. (2016). SampleRNN: An unconditional end-to-end neural audio generation model. arXiv:1612.07837.

Mehta, H., Gupta, A., Cutkosky, A., & Neyshabur, B. (2022). Long range language modeling via gated state spaces. arXiv:2206.13947.

Meloni, E., Betti, A., Faggi, L., Marullo, S., Tiezzi, M., & Melacci, S. (2021). Evaluating continual learning algorithms by generating 3d virtual environments. In *International workshop on continual semi-supervised learning* (pp. 62–74). Springer.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., & Graves, A. (2020). A practical sparse approximation for real time recurrent learning. arXiv:2006.07232.

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. arXiv:1609.07843.

Merrill, W., Petty, J., & Sabharwal, A. (2024). The illusion of state in state-space models. In *Forty-first international conference on machine learning*.

Merrill, W., & Sabharwal, A. (2023a). A logic for expressing log-precision transformers. *Advances in Neural Information Processing Systems*, *36*, 52453–52463.

Merrill, W., & Sabharwal, A. (2023b). The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, *11*, 531–545 https://doi.org/10.1162/tacl_a_00562

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, *20*(3), 498–511. https://doi.org/10.1109/TNN.2008.2010350

Micheli, A., & Tortorella, D. (2022). Discrete-time dynamic graph echo state networks. *Neurocomputing*, *496*, 85–95. https://doi.org/10.1016/j.neucom.2022.05.001

Mihaylov, T., Clark, P., Khot, T., & Sabharwal, A. (2018). Can a suit of armor conduct electricity? A new dataset for open book question answering. arXiv:1809.02789.

Mikolov, T., & Zweig, G. (2012). Context dependent recurrent neural network language model. In *2012 IEEE spoken language technology workshop (SLT)* (pp. 234–239). IEEE.

Millidge, B., Tschantz, A., & Buckley, C. L. (2022). Predictive coding approximates backprop along arbitrary computation graphs. *Neural Computation*, *34*(6), 1329–1368. https://doi.org/10.1162/neco_a_01497

Mori, D. et al. (1989). Bps: A learning algorithm for capturing the dynamic nature of speech. In *International 1989 joint conference on neural networks* (pp. 417–423). IEEE.

Morrill, J., Kidger, P., Yang, L., & Lyons, T. (2021). Neural controlled differential equations for online prediction tasks. arXiv:2106.11028.

Mozer, M. C. (2013). A focused backpropagation algorithm for temporal pattern recognition. In *Backpropagation* (pp. 137–169). Psychology Press.

Munkhdalai, T., Faruqui, M., & Gopal, S. (2024). Leave no context behind: Efficient infinite context transformers with infini-attention. arXiv:2404.07143.

Nangia, N., & Bowman, S. R. (2018). Listops: A diagnostic dataset for latent tree learning. arXiv:1804.06028.

Nguyen, E., Poli, M., Faizi, M., Thomas, A., Wornow, M., Birch-Sykes, C., Massaroli, S., Patel, A., Rabideau, C., Bengio, Y. et al. (2024). Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *Advances in Neural Information Processing Systems*, *36*.

Norrie, T., Patil, N., Yoon, D. H., Kurian, G., Li, S., Laudon, J., Young, C., Jouppi, N., & Patterson, D. (2021). The design process for Google's training chips: TPUv2 and TPUv3. *IEEE Micro*, *41*(2), 56–63.

Oliva, J., Neiswanger, W., Póczos, B., Xing, E., Trac, H., Ho, S., & Schneider, J. (2015). Fast function to function regression. In *Artificial intelligence and statistics* (pp. 717–725). PMLR.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A. et al. (2022). In-context learning and induction heads. arXiv:2209.11895.

Oren, M., Hassid, M., Adi, Y., & Schwartz, R. (2024). Transformers are multi-state rnns. arXiv:2401.06104.

Ororbia, A., Mali, A., Giles, C. L., & Kifer, D. (2020). Continual learning of recurrent neural networks by locally aligning distributed representations. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(10), 4267–4278.

Ororbia, A. G. (2023). Brain-inspired machine intelligence: A survey of neurobiologically-plausible credit assignment. arXiv:2312.09257.

Ororbia, A. G., & Mali, A. (2019). Biologically motivated algorithms for propagating local target representations. In *Proceedings of the AAAI conference on artificial intelligence* (pp. 4651–4658). (*vol. 33*).

Ororbia, A. G., Mali, A., Kifer, D., & Giles, C. L. (2018). Deep credit assignment by aligning local representations. arXiv:1803.01834.

Orvieto, A., De, S., Gulcehre, C., Pascanu, R., & Smith, S. L. (2024). Universality of linear recurrences followed by non-linear projections: Finite-width guarantees and benefits of complex eigenvalues. In *Forty-first international conference on machine learning*.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., & De, S. (2023). Resurrecting recurrent neural networks for long sequences. In *International conference on machine learning* (pp. 26670–26698). PMLR.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N.-Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., & Fernández, R. (2016). The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1525–1534).

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318).

Parisi, G. I., & Lomonaco, V. (2020). Online continual learning on sequences. In *Recent trends in learning from data: Tutorials from the INNS big data and deep learning conference (INNSBDDL2019)* (pp. 197–221). Springer.

Park, I. M., Ságodi, Á., & Sokół, P. A. (2023). Persistent learning signals and working memory without continuous attractors. arXiv:2308.12585.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, Ł., Shazeer, N., Ku, A., & Tran, D. (2018). Image transformer. arXiv:1802.05751.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318). Pmlr.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, *1*(2), 263–269 https://doi.org/10.1162/NECO.1989.1.2.263

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228. https://doi.org/10.1109/72.410363

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K. et al. (2023). Rwkv: Reinventing rnns for the transformer era. arXiv:2305.13048.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Du, X., Ferdinan, T., Hou, H., Kazienko, P., Kiran GV, K., Kocoń, J., Koptyra, B., Krishna, S., McClelland Jr., R., Lin, J., Muennighoff, N., Obeid, F. et al. (2024a). Eagle and finch: RWKV with matrix-valued states and dynamic recurrence. arXiv:2404.05892.

Peng, B., Narayanan, S., & Papadimitriou, C. (2024b). On limitations of the transformer architecture. In *First conference on language modeling*.

Peng, B., Zhang, R., Goldstein, D., Alcaide, E., Du, X., Hou, H., Lin, J., Liu, J., Lu, J., Merrill, W. et al. (2025). Rwkv-7" goose" with expressive dynamic state evolution. arXiv:2503.14456.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N., & Kong, L. (2021). Random feature attention. In *International conference on learning representations*. https://openreview.net/forum?id=QtTKTdVrFBB.

Pérez, J., Marinković, J., & Barceló, P. (2019). On the turing completeness of modern neural network architectures. arXiv:1901.03429.

Pilault, J., Fathi, M., Firat, O., Pal, C., Bacon, P.-L., & Goroshin, R. (2023). Block-state transformers. In *Thirty-seventh conference on neural information processing systems*.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, *59*(19), 2229.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., & Ré, C. (2023). Hyena hierarchy: Towards larger convolutional language models. In *International conference on machine learning, ICML 2023, 23–29 July 2023, Honolulu, Hawaii, USA* (pp. 28043–28078). PMLR (*vol. 202*). Proceedings of Machine Learning Research. https://proceedings.mlr.press/v202/poli23a.html.

Poli, M., Thomas, A. W., Nguyen, E., Ponnusamy, P., Deiseroth, B., Kersting, K., Suzuki, T., Hie, B., Ermon, S., Re, C. et al. (2024). Mechanistic design and scaling of hybrid architectures. In *Forty-first international conference on machine learning*.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., & Dean, J. (2023). Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, *5*.

Pramanik, S., Elelimy, E., Machado, M. C., & White, A. (2023). Recurrent linear transformers.

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., & Zhong, Y. (2022a). The devil in linear transformer. In *Proceedings of the 2022 conference on empirical methods in natural language processing* (pp. 7025–7041).

Qin, Z., Li, D., Sun, W., Sun, W., Shen, X., Han, X., Wei, Y., Lv, B., Yuan, F., Luo, X. et al. (2023a). Scaling transnormer to 175 billion parameters. arXiv:2307.14995.

Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., & Zhong, Y. (2022b). Cosformer: Rethinking softmax in attention. In *The tenth international conference on learning representations, ICLR 2022, Virtual Event, April 25–29, 2022*. OpenReview.net. https://openreview.net/forum?id=Bl8CQrx2Up4.

Qin, Z., Sun, W., Li, D., Shen, X., Sun, W., & Zhong, Y. (2024). Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. arXiv:2401.04658.

Qin, Z., Sun, W., Lu, K., Deng, H., Li, D., Han, X., Dai, Y., Kong, L., & Zhong, Y. (2023b). Linearized relative positional encoding. arXiv:2307.09270.

Qin, Z., Yang, S., & Zhong, Y. (2023c). Hierarchically gated recurrent neural network for sequence modeling. In *Thirty-seventh conference on neural information processing systems*.

Radev, D. R., Muthukrishnan, P., Qazvinian, V., & Abu-Jbara, A. (2013). The ACL anthology network corpus. *Language Resources and Evaluation*, *47*(4), 919–944.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., & Lillicrap, T. P. (2019). Compressive transformers for long-range sequence modelling. In *International conference on learning representations*.

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, *20*.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv:1710.05941.

Ravi, S., & Larochelle, H. (2016). Optimization as a model for few-shot learning. In *International conference on learning representations*.

Rawat, A. S., Chen, J., Yu, F. X. X., Suresh, A. T., & Kumar, S. (2019). Sampled softmax with random fourier features. *Advances in Neural Information Processing Systems*, *32*.

Reyna, M. A., Josef, C. S., Jeter, R., Shashikumar, S. P., Westover, M. B., Nemati, S., Clifford, G. D., & Sharma, A. (2020). Early prediction of sepsis from clinical data: The physionet/computing in cardiology challenge 2019. *Critical Care Medicine*, *48*(2), 210–217.

Romero, D. W., Kuzina, A., Bekkers, E. J., Tomczak, J. M., & Hoogendoorn, M. (2021). Ckconv: Continuous kernel convolution for sequential data. In *International conference on learning representations*.

Rubanova, Y., Chen, R. T. Q., & Duvenaud, D. K. (2019). Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems*, *32*.

Rudin, W. et al. (1976). Principles of mathematical analysis (vol. 3). McGraw-hill New York.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation.

Rusch, T. K., & Mishra, S. (2020). Coupled oscillatory recurrent neural network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies. In *International conference on learning representations*.

Rusch, T. K., & Mishra, S. (2021). Unicornn: A recurrent model for learning very long time dependencies. In *International conference on machine learning* (pp. 9168–9178). PMLR.

Rusch, T. K., Mishra, S., Erichson, N. B., & Mahoney, M. W. (2021). Long expressive memory for sequence modeling. In *International conference on learning representations*.

Ryoo, M. S., Gopalakrishnan, K., Kahatapitiya, K., Xiao, T., Rao, K., Stone, A., Lu, Y., Ibarz, J., & Arnab, A. (2023). Token turing machines. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 19070–19081).

Sakaguchi, K., Bras, R. L., Bhagavatula, C., & Choi, Y. (2021). Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, *64*(9), 99–106.

Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent advances in recurrent neural networks. arXiv:1801.01078.

Scardapane, S., Gallicchio, C., Micheli, A., & Soriano, M. C. (2023). Guest editorial: Trends in reservoir computing. *Cognitive Computation*, *15*(5), 1407–1408. https://doi.org/10.1007/s12559-021-09890-1

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Schäfer, A. M., & Zimmermann, H. G. (2006). Recurrent neural networks are universal approximators. In *Artificial neural networks–ICANN 2006: 16th international conference, Athens, Greece, September 10–14, 2006. proceedings, part I 16* (pp. 632–640). Springer.

Schirmer, M., Eltayeb, M., Lessmann, S., & Rudolph, M. (2022). Modeling irregular time series with continuous recurrent units. In *International conference on machine learning* (pp. 19388–19405). PMLR.

Schlag, I., Irie, K., & Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers. In *International conference on machine learning* (pp. 9355–9366). PMLR.

Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, *4*(1), 131–139.

Schmidhuber, J. (1993). A 'self-referential'weight matrix. In *ICANN'93: Proceedings of the international conference on artificial neural networks Amsterdam, the Netherlands 13–16 September 1993 3* (pp. 446–450). Springer.

Schmidhuber, J. (AI Blog, 2021). 26 March 1991: Neural nets learn to program neural nets with fast weights–like today's Transformer variants. 2021: New stuff! https://people.idsia.ch/juergen/fast-weight-programmer-1991-transformer.html.

Selva, J., Johansen, A. S., Escalera, S., Nasrollahi, K., Moeslund, T. B., & Clapés, A. (2023). Video transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Shahroudy, A., Liu, J., Ng, T.-T., & Wang, G. (2016). Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1010–1019).

Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2018). Ordered neurons: Integrating tree structures into recurrent neural networks. In *International conference on learning representations*.

Sieber, J., Alonso, C. A., Didier, A., Zeilinger, M. N., & Orvieto, A. (2024). Understanding the differences in foundation models: attention, state space models, and recurrent neural networks. arXiv:2405.15731.

Siegelmann, H. T. (2012). Neural networks and analog computation: Beyond the turing limit. Springer Science & Business Media.

Siegelmann, H. T., & Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 440–449).

Siems, J., Carstensen, T., Zela, A., Hutter, F., Pontil, M., & Grazzi, R. (2025). Deltaproduct: Improving state-tracking in linear RNNs via householder products. arXiv:2502.10297.

Sigurdsson, G. A., Varol, G., Wang, X., Farhadi, A., Laptev, I., & Gupta, A. (2016). Hollywood in homes: Crowdsourcing data collection for activity understanding. In *Computer vision–ECCV 2016: 14th European conference, Amsterdam, the Netherlands, October 11–14, 2016, proceedings, part i 14* (pp. 510–526). Springer.

Simon, H. A., & Kotovsky, K. (1963). Human acquisition of concepts for sequential patterns. *Psychological Review*, *70*(6), 534.

Smith, J. T. H., Warrington, A., & Linderman, S. W. (2022). Simplified state space layers for sequence modeling. arXiv:2208.04933.

So, D., Mańke, W., Liu, H., Dai, Z., Shazeer, N., & Le, Q. V. (2021). Searching for efficient transformers for language modeling. *Advances in Neural Information Processing Systems*, *34*, 6010–6022.

Sodhani, S., Chandar, S., & Bengio, Y. (2020). Toward training recurrent neural networks for lifelong learning. *Neural Computation*, *32*(1), 1–35.

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3), 714–735.

Srivastava, N., Mansimov, E., & Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International conference on machine learning* (pp. 843–852). PMLR.

Stahlberg, F. (2020). Neural machine translation: A review. *Journal of Artificial Intelligence Research*, *69*, 343–418.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., & Wei, F. (2023a). Retentive network: A successor to transformer for large language models. arXiv:2307.08621.

Sun, Y., Dong, L., Patra, B., Ma, S., Huang, S., Benhaim, A., Chaudhary, V., Song, X., & Wei, F. (2023b). A length-extrapolatable transformer. In A. Rogers, J. L. Boyd-Graber, & N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers), ACL 2023, Toronto, Canada, July 9–14, 2023* (pp. 14590–14604). Association for Computational Linguistics https://doi.org/10.18653/V1/2023.ACL-LONG.816

Tallec, C., & Ollivier, Y. (2017). Unbiased online recurrent optimization. arXiv:1702.05043.

Tallec, C., & Ollivier, Y. (2018). Can recurrent neural networks warp time? In *International conference on learning representation 2018*.

Tan, C. W., Bergmeir, C., Petitjean, F., & Webb, G. I. (2020). Monash university, uea, ucr time series regression archive. arXiv:2006.10996.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., & Metzler, D. (2020). Long range arena: A benchmark for efficient transformers. arXiv:2011.04006.

Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2023). Efficient transformers: A survey. *ACM Computing Surveys*, *55*(6), 109:1–109:28. https://doi.org/10.1145/3530811

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., & Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning* (pp. 2722–2731). PMLR.

Team, J., Lenz, B., Arazi, A., Bergman, A., Manevich, A., Peleg, B., Aviram, B., Almagor, C., Fridman, C., Padnos, D. et al. (2024). Jamba-1.5: Hybrid transformer-mamba models at scale. arXiv:2408.12570.

Tiezzi, M., Marra, G., Melacci, S., & Maggini, M. (2021). Deep constraint-based propagation in graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(2), 727–739.

Tiezzi, M., Marra, G., Melacci, S., Maggini, M., & Gori, M. (2020a). A lagrangian approach to information propagation in graph neural networks. In *ECAI 2020* (pp. 1539–1546). IOS Press.

Tiezzi, M., Marullo, S., Faggi, L., Meloni, E., Betti, A., & Melacci, S. (2022). Stochastic coherence over attention trajectory for continuous learning in video streams. In L. De Raedt (Ed.), *Proceedings of the thirty-first international joint conference on artificial intelligence, IJCAI-22* (pp. 3480–3486). International Joint Conferences on Artificial Intelligence Organization. Main Track. https://doi.org/10.24963/ijcai.2022/483

Tiezzi, M., Melacci, S., Betti, A., Maggini, M., & Gori, M. (2020b). Focus of attention improves information transfer in visual features. *Advances in Neural Information Processing Systems*, *33*, 22194–22204.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. In *International conference on machine learning* (pp. 10347–10357). PMLR.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., & Salakhutdinov, R. (2019). Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 4344–4353).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D. et al. (2016). Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, *29*.

Voelker, A., Kajić, I., & Eliasmith, C. (2019). Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in Neural Information Processing Systems*, .

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International conference on learning representations*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. arXiv:2006.04768.

Wang, Z., Ma, Y., Liu, Z., & Tang, J. (2019). R-transformer: Recurrent neural network enhanced transformer. arXiv:1907.05572.

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. arXiv:1804.03209.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, *1*(4), 339–356.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Whittington, J. C. R., & Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, *29*(5), 1229–1262.

Widrow, B., Hoff, M. E. et al. (1960). Adaptive switching circuits. In *IRE WESCON convention record* (pp. 96–104). New York (*vol. 4*).

Williams, R. J., & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, *2*(4), 490–501.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280.

Wu, Q., Lan, Z., Qian, K., Gu, J., Geramifard, A., & Yu, Z. (2022a). Memformer: A memory-augmented transformer for sequence modeling. In *Findings of the association for computational linguistics: AACL-IJCNLP 2022* (pp. 308–318).

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 3–19).

Wu, Y., Rabe, M. N., Hutchins, D., & Szegedy, C. (2022b). Memorizing transformers. arXiv:2203.08913.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks And Learning Systems*, *32*(1), 4–24.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747.

Yang, S., Kautz, J., & Hatamizadeh, A. (2025). Gated delta networks: Improving mamba2 with delta rule. arXiv:2412.06464.

Yang, S., Wang, B., Shen, Y., Panda, R., & Kim, Y. (2023). Gated linear attention transformers with hardware-efficient training.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, *31*(7), 1235–1270.

Zellers, R., Bisk, Y., Schwartz, R., & Choi, Y. (2018). Swag: A large-scale adversarial dataset for grounded commonsense inference. arXiv:1808.05326.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 4791–4800).

Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2023). Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence* (pp. 11121–11128). (*vol. 37*).

Zhai, S., Talbott, W., Srivastava, N., Huang, C., Goh, H., Zhang, R., & Susskind, J. (2021). An attention free transformer. arXiv:2105.14103.

Zhang, M., Saab, K. K., Poli, M., Dao, T., Goel, K., & Ré, C. (2023). Effectively modeling time series with simple discrete state spaces. In *The eleventh international conference on learning representations, ICLR 2023, Kigali, Rwanda, May 1–5, 2023*. OpenReview.net. https://openreview.net/pdf?id=2EpjkjzdCAa.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z. et al. (2023). A survey of large language models. arXiv:2303.18223.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision* (pp. 19–27).

Zhuang, B., Liu, J., Pan, Z., He, H., Weng, Y., & Shen, C. (2023). A survey on efficient training of transformers. arXiv:2302.01107.

Zucchet, N., Kobayashi, S., Akram, Y., von Oswald, J., Larcher, M., Steger, A., & Sacramento, J. (2023a). Gated recurrent neural networks discover attention. arXiv:2309.01775.

Zucchet, N., Meier, R., Schug, S., Mujika, A., & Sacramento, J. (2023b). Online learning of long-range dependencies. *Advances in Neural Information Processing Systems*, *36*, 10477–10493.