Tensor Networks for Probabilistic Sequence Modeling

Jacob Miller
Mila and DIRO
Université de Montréal
Montréal QC, Canada
jmjacobmiller@gmail.com

Guillaume Rabusseau
Mila and DIRO
Université de Montréal
Montréal QC, Canada
grabus@iro.umontreal.ca

John Terilla
CUNY and Tunnel
City University of New York
New York NY, USA
jterilla@gc.cuny.edu

Abstract

Tensor networks are a powerful modeling framework developed for computational many-body physics, which have only recently been applied within machine learning. In this work we utilize a uniform matrix product state (u-MPS) model for probabilistic modeling of sequence data. We first show that u-MPS enable sequence-level parallelism, with length-n sequences able to be evaluated in depth $O(\log n)$. We then introduce a novel generative algorithm giving trained u-MPS the ability to efficiently sample from a wide variety of conditional distributions, each one defined by a regular expression. Special cases of this algorithm correspond to autoregressive and fill-in-the-blank sampling, but more complex regular expressions permit the generation of richly structured text in a manner that has no direct analogue in current generative models. Experiments on synthetic text data find u-MPS outperforming LSTM baselines in several sampling tasks, and demonstrate strong generalization in the presence of limited data.

1 Introduction

Tensor network models have long represented the state of the art in modeling complex quantum systems [39, 13, 26], but have only recently been utilized as models for machine learning [24, 10, 36, 25, 19, 35, 8]. In contrast to neural networks, tensor networks forgo the use of nonlinear activation functions, relying instead on multiplicative interactions to capture complex correlations within data. This gives tensor networks a convenient mathematical structure suitable for proving powerful theoretical results, such as the separation in expressivity between almost all deep tensor networks and their shallow counterparts [10]. However, these distinctive properties have yet to be leveraged for attaining equally impressive *operational* capabilities, which would give support for the wider adoption of tensor network models in real-world machine learning tasks.

In this work we apply a recurrent tensor network, the *uniform matrix product state* (u-MPS), to the task of probabilistic sequence modeling, and identify several novel abilities of u-MPS regarding their evaluation and generative capabilities. Despite its recurrent nature, we show that sequential inputs to u-MPS can be processed in a highly parallel manner, with sequences of length n being evaluated in parallel time $\mathcal{O}(\log n)$. While the difficulty of parallelizing deep recurrent neural networks (RNNs) has previously motivated the development of non-recurrent architectures for sequence processing tasks (e.g. [17, 38]), our finding shows that recurrent tensor networks represent another means of achieving greater parallelism.

We further show that u-MPS models are endowed with surprising generative capabilities closely tied to the structure of regular expressions (regex). While standard autoregressive models are constrained to generate sequences in a stream-like fashion, we find that u-MPS permit many different forms of sampling, which are in one-to-one correspondence with regular expressions R. Our sampling algorithm efficiently produces unbiased samples from the probability distribution learned by the u-MPS, conditioned on the output sequence matching a given regular expression R.

Preprint. Under review.

For example, letting Σ^* denote regex matching all sequences over an alphabet Σ, and p, s a given prefix and suffix, the choices $R = \Sigma^*$ and $R = p\Sigma^* s$ respectively generate standard autoregressive-style sampling and fill-in-the-blank sampling, where a missing subsequence is inferred from the bidirectional context of p and s. Sampling with more general regex permits the generation of sequences with rich internal structure, a capability with particular promise for many practical tasks (e.g., automatic code generation). Experiments on several synthetic text datasets show strong generalization capabilities, with the u-MPS able to successfully infer the structure of strings of significantly longer length than those used for training.

Summary of Contributions We give the first implementation of a u-MPS in probabilistic sequence modeling, and identify several surprising properties of this model. The absence of nonlinear activation functions in the u-MPS allows us to utilize a parallel evaluation method during training and inference. We also introduce a flexible recursive sampling algorithm for the u-MPS whose capabilities generalize those of essentially all sampling methods based on neural networks. We expect these contributions to open significant new research directions in the design of sequential generative models, with language modeling being a particularly promising domain.

Related Work Notable previous applications of tensor networks in machine learning include compressing large neural network weights [24], proving separations in the expressivity of deep vs shallow networks [10], and for supervised [36, 25, 18] and unsupervised [19, 35, 8] learning tasks. Of particular relevance is [34], where (non-uniform) MPS were trained as generative models for fixed-length binary sequences using the density matrix renormalization group (DMRG) algorithm. This work can be seen as a continuation of [30], where u-MPS were introduced from a theoretical perspective as a language model, but without the parallelization, sampling, or experimental results given here. Our sampling algorithm is a significant generalization of the fixed-length algorithm introduced in [19] (which in turn follows that of [14]), and by virtue of the recurrent nature of u-MPS, permits the generation of discrete sequences of arbitrary length. The completely positive maps employed in our sampling algorithm are similar to those used within hidden quantum Markov models [22, 33], and likewise admit a natural interpretation in terms of concepts from quantum information theory.

Models equivalent to u-MPS have been proposed as a quadratic generalization of weighted finite automata (WFA) [2] (see also [3] for similar methods). u-MPS can be seen as a particular case of linear second-order RNNs, whose connections with WFA were explored in [32]. The benefits of linear RNNs for parallelization and interpretability were studied in [21, 15]. A key difference from these prior works is our use of u-MPS for complex sampling tasks.

Finally, there have been a number of theoretical proposals for the use of different tensor network architectures for modeling and understanding natural language, such as [31, 9, 16, 11]. Our work demonstrate that such models are not just of theoretical interest, but can have compelling practical benefits as well.

2 Background

We consider sequences over a finite alphabet Σ, with Σ^n denoting the set of all length-n strings, Σ^* the set of all strings, and ε the empty string. We use $\|v\|$ to denote the 2-norm of a vector, matrix, or higher-order tensor v, and $\mathrm{Tr}(M) = \sum_{i=1}^{D} M_{ii}$ to denote the trace of a square matrix $M \in \mathbb{R}^{D\times D}$.

A real-valued[1] tensor $\mathcal{T} \in \mathbb{R}^{d_1\times d_2\times\cdots\times d_n}$ is said to have shape $(d_1, d_2, \ldots, d_n)$, and can be specified by an indexed collection of elements $\mathcal{T}_{i_1,i_2,\ldots,i_n} \in \mathbb{R}$, where each index $i_k \in [d_k] := \{1, 2, \ldots, d_k\}$. Tensors with n indices are said to be nth order, and the set of nth order tensors form a vector space of dimension $\Pi_{k=1}^{n} d_k$. Matrices, vectors, and scalars are the simplest examples of tensors, of 2nd, 1st, and 0th order, respectively. Tensor contraction is a generalization of both matrix multiplication and vector inner product, and multiplies two tensors along a pair of indices with equal dimension. If the tensors $\mathcal{T}$ and $\mathcal{T}'$ have respective shapes $(d_1, \ldots, d_k, \ldots, d_n)$ and $(d'_1, \ldots, d'_{k'}, \ldots, d'_{n'})$, for $d_k = d'_{k'}$, then the contraction of the k and k' indices gives a product tensor $\mathcal{T}''$, described by

[1] The restriction to real-valued tensors is natural for machine learning, but differs from the standard in quantum physics of using complex parameters. The results given here carry over to the complex setting, and only require the replacement of some tensors by their complex conjugate.

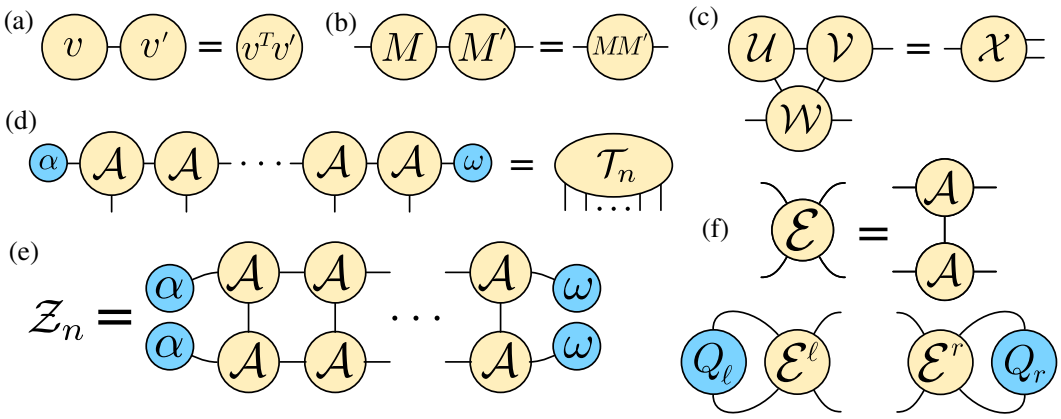

Figure 1: (a-b) Two well-known cases of tensor contractions, inner products of vectors and matrix multiplication. (c) A simple tensor network, where 2nd, 3rd, and 4th order tensors are contracted to form a 3rd order tensor. In numerical libraries, small tensor contractions can be computed with the `einsum` function, and the output $\mathcal{X}$ is independent of contraction order. (d) The u-MPS model, which uses a core tensor $\mathcal{A}$ of shape (D, d, D) and D-dimensional vectors α and ω to define tensors of arbitrary order. (e) The length-n normalization factor $\mathcal{Z}_n$ defined by (3), expressed as a network of tensor contractions. (f) The 4th order tensor $\mathcal{E}$ defined by two copies of the u-MPS core tensor $\mathcal{A}$. The contraction of $\mathcal{E}$ with a matrix on the left or right gives the left and right *transfer operators* of the u-MPS, linear maps which allow the efficient computation of $\mathcal{Z}_n$ via (4).

elements

$$T''_{i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_n,i'_1,\ldots,i'_{k'-1},i'_{k'+1},\ldots,i'_{n'}} = \sum_{i_k=1}^{d_k} \mathcal{T}_{i_1,\ldots,i_k,\ldots,i_n} \mathcal{T}'_{i'_1,\ldots,i_k,\ldots,i'_{n'}}. \tag{1}$$

The contraction operation (1) is more easily understood with a convenient graphical notation (see Figure 1), where individual tensors correspond to nodes in an undirected graph, and edges describe contractions to be performed. Contracting along an index corresponds to merging two connected nodes, to produce a new node whose outgoing edges are the union of those in the tensors being contracted. An important property of tensor contraction is its generalized associativity, so that a network of tensors can be contracted in any order, with the final product tensor being the same in every case.

A natural example of an nth order tensor is a probability distribution over length-n sequences Σ^n, where the probabilities associated with all possible sequences form the $|\Sigma|^n$ separate tensor elements. This exponential growth in the number of elements makes dense representations of higher order tensors infeasible, but convenient tensor decompositions frequently permit the efficient manipulation of tensors with high order, even into the thousands.

The fixed-size matrix product state [29] (MPS, also known as tensor train [27]) model parameterizes an nth order tensor $\mathcal{T}$ with shape $(d_1, d_2, \ldots, d_n)$ as a sequential contraction of n independent tensor "cores" $\{\mathcal{A}^{(j)}\}_{j=1}^n$, which form the parameters of the model. Each $\mathcal{A}^{(j)}$ has shape (D_{j-1}, d_j, D_j), where $D_0 = D_n = 1$. The dimensions D_j are referred to as bond dimensions (or ranks) of the MPS, and by choosing the D_j to be sufficiently large, it is possible to exactly represent any nth order tensor.

3 Uniform MPS

In this work we utilize the *uniform MPS* (u-MPS) model, a recurrent tensor network obtained by choosing all cores of an MPS to be identical tensors $\mathcal{A}^{(j)} = \mathcal{A}$ with shape (D, d, D). To obtain scalar tensor elements, D-dimensional vectors α and ω are used as "boundary conditions" to terminate the initial and final bond dimensions of the network. In contrast to fixed-length MPS, the recurrent nature

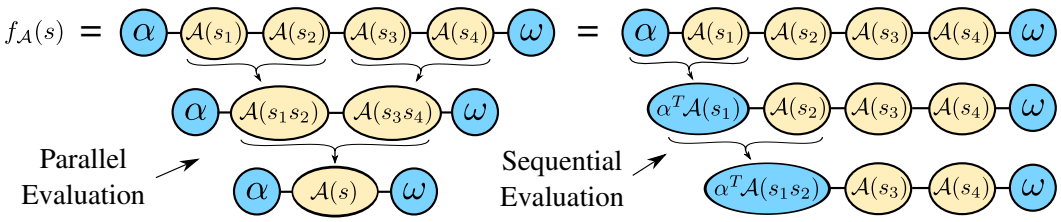

Figure 2: Illustration of parallel and sequential evaluation of $f_{\mathcal{A}}(s)$ when $|s| = 4$, where $f_{\mathcal{A}}(s) = (\mathcal{T}_4)_{i_1,i_2,i_3,i_4}$, an element of the 4th order tensor defined by a u-MPS. After obtaining the matrix representations $\mathcal{A}(s_1), \ldots, \mathcal{A}(s_n)$ from s, parallel evaluation involves repeated batch multiplications of nearest-neighbor pairs of matrices, with the boundary vectors α and ω only incorporated after the matrix product $\mathcal{A}(s)$ has been obtained. Sequential evaluation instead uses iterated matrix-vector multiplications starting with a boundary vector to contract this product. Parallel and sequential evaluation have respective costs of $\mathcal{O}(nD^3)$ and $\mathcal{O}(nD^2)$, but the former can be carried out in $\mathcal{O}(\log n)$ parallel time. The mathematical equivalence of these evaluation strategies is a basic example of the associativity of tensor contractions, allowing an appropriate method to be chosen based on the size of the model, the problem at hand, and the availability of hardware acceleration.

of u-MPS allows the generation of nth order tensors $\mathcal{T}_n \in \mathbb{R}^{d^n}$ for any $n \in \mathbb{N}$, which in turn allows u-MPS to be applied in problems involving sequential data.

For discrete sequences over an alphabet Σ of size d, a u-MPS (paired with a bijection $\varphi : \Sigma \to [d]$) can be used to map a sequence of arbitrary length-n to the index of an nth order tensor $\mathcal{T}_n$, defining a scalar-valued function $f_{\mathcal{A}}$ over sequences. Using $\mathcal{A}(c) = \mathcal{A}_{:,\varphi(c),:} \in \mathbb{R}^{D \times D}$ to denote the matrix associated with the character $c \in \Sigma$, a u-MPS acts on a sequence $s = s_1 s_2 \cdots s_n \in \Sigma^n$ as

$$f_{\mathcal{A}}(s) = \alpha^T \mathcal{A}(s_1)\mathcal{A}(s_2) \cdots \mathcal{A}(s_n)\omega = \alpha^T \mathcal{A}(s)\omega, \tag{2}$$

where we use $\mathcal{A}(s) := \mathcal{A}(s_1)\mathcal{A}(s_2) \cdots \mathcal{A}(s_n)$ to denote the matrix product appearing in (2). The function $\mathcal{A}(s)$ can be seen as a matrix-valued representation of arbitrary sequences $s \in \Sigma^*$, and is *compositional* in the sense that st is represented by the product of representations $\mathcal{A}(s)$ and $\mathcal{A}(t)$.

While u-MPS are clearly laid out as a sequential model, the evaluation of $f_{\mathcal{A}}(s)$ for $|s| = n$ can be parallelized by evaluating (2) using $\lceil \log_2(n) \rceil$ batched matrix-matrix multiplications on all nearest-neighbor pairs of matrices, as shown in Figure 2. This form of parallelization requires the absence of nonlinear activation functions in the evaluation, and can also be carried out in linear RNNs [21].

3.1 Born Machines

While (2) is identical to the evaluation rule for WFA, and well-suited for regression tasks, we are interested in using u-MPS as probabilistic models. This requires the interpretation of $f_{\mathcal{A}}(s)$ as a non-negative probability $P(s)$, and deciding if a general WFA outputs negative values is undecidable [12]. This issue can be circumvented by requiring all entries of $\mathcal{A}$, α, and ω to be non-negative real numbers, but such models can be seen as largely equivalent to hidden Markov models [12].

We instead follow the approach introduced in [30] (see also [19]), which is inspired by the typical usage of MPS in quantum mechanics. For the case of u-MPS, this *Born machine* approach converts a scalar value $f_{\mathcal{A}}(s)$ to an unnormalized probability $\tilde{P}(s) := |f_{\mathcal{A}}(s)|^2$. This can be converted into a properly normalized distribution over sequence of fixed length n by choosing $P_n(s) = \tilde{P}(s)/\mathcal{Z}_n$, where the normalization function $\mathcal{Z}_n$ is given by

$$\mathcal{Z}_n = \sum_{s \in \Sigma_n} \tilde{P}(s) = \sum_{i_1 \in [d]} \sum_{i_2 \in [d]} \cdots \sum_{i_n \in [d]} |(T_n)_{i_1,i_2,\ldots,i_d}|^2 = \|\mathcal{T}_n\|^2, \tag{3}$$

and with $\mathcal{T}_n$ the nth order tensor defined by the u-MPS. This quadratic evaluation rule is equivalent to the Born rule of quantum mechanics [6], which gives a formal interpretation of such models as wavefunctions over n quantum spins. However this probabilistic correspondence is richer in the case of u-MPS, since distributions over sequences of different lengths can be easily defined. The distribution $P_*(s) = \tilde{P}(s)/\mathcal{Z}_*$ in particular gives a probability distribution over strings of arbitrary length, where the normalization factor $\mathcal{Z}_*$ is identical to that given in (3), but with the sum over Σ^n

Table 1: Dictionary giving the correspondence between regular expressions (regex) and generalized transfer operators associated with a u-MPS (note the reversal of order in $\mathcal{E}^{\ell}_{R_1R_2}$). The positive semidefinite matrix Q^*_r is defined in terms of an infinite sum, but can also be computed as the solution to the linear equation $(I - \mathcal{E}^r_S)Q^*_r = Q_r$ (similarly for Q^*_ℓ).

REGEX R =	c	R_1R_2	$R_1\|R_2$	S^*
$\boldsymbol{\mathcal{E}^r_R(Q_r)}$ =	$\mathcal{A}_c Q_r \mathcal{A}_c^T$	$\mathcal{E}^r_{R_1}(\mathcal{E}^r_{R_2}(Q_r))$	$\mathcal{E}^r_{R_1}(Q_r) + \mathcal{E}^r_{R_2}(Q_r)$	$\sum_{n=0}^{\infty}(\mathcal{E}^r_S)^{\circ n}(Q_r) =: Q^*_r$
$\boldsymbol{\mathcal{E}^\ell_R(Q_\ell)}$ =	$\mathcal{A}_c^T Q_\ell \mathcal{A}_c$	$\mathcal{E}^\ell_{R_2}(\mathcal{E}^\ell_{R_1}(Q_\ell))$	$\mathcal{E}^\ell_{R_1}(Q_\ell) + \mathcal{E}^\ell_{R_2}(Q_\ell)$	$\sum_{n=0}^{\infty}(\mathcal{E}^\ell_S)^{\circ n}(Q_\ell) =: Q^*_\ell$

replaced by one over Σ^* (assuming this sum converges). We show in Section 4 how normalization functions of this form can be generalized further to incorporate sums over all strings matching an arbitrary regular expression R.

Normalization functions like $\mathcal{Z}_n$ occur frequently in many-body physics, and can be efficiently computed via a simple reordering of tensor contractions. By (3), $\mathcal{Z}_n$ equals the 2-norm of $\mathcal{T}_n$, which is represented diagrammatically as Figure 1e. The naive method of evaluating $\mathcal{Z}_n$ involves first generating all elements of $\mathcal{T}_n$ via contraction along the horizontal D-dimensional indices of the u-MPS, but the generalized associativity of tensor contraction lets us evaluate this expression more efficiently.

By first contracting two copies of $\mathcal{A}$ along a vertical d-dimensional index (see (1)f) we obtain a 4th order tensor $\mathcal{E}$, which can be interpreted as a linear map on a space of matrices in two main ways, by contracting either its left or its right indices with an input. These linear maps, known as *transfer operators*, are examples of completely positive (CP) maps, a generalization of stochastic matrices which find frequent application in the context of quantum information theory (see Appendix A for more details). These maps admit the Kraus representations $\mathcal{E}^r(Q_r) = \sum_{c\in\Sigma}\mathcal{A}(c)Q_r\mathcal{A}(c)^T$ and $\mathcal{E}^\ell(Q_\ell) = \sum_{c\in\Sigma}\mathcal{A}(c)^T Q_\ell \mathcal{A}(c)$, which are connected by the adjoint identity $\mathrm{Tr}(Q_\ell \mathcal{E}^r(Q_r)) = \mathrm{Tr}(\mathcal{E}^\ell(Q_\ell)Q_r)$.[2]

The normalization $\mathcal{Z}_n$ can be equivalently computed in terms of left or right transfer operators, with the latter option yielding

$$\mathcal{Z}_n = \alpha^T \mathcal{E}^r(\mathcal{E}^r(\cdots\mathcal{E}^r(\omega\omega^T))\cdots)\alpha = \mathrm{Tr}\left(Q^\alpha_\ell\ (\mathcal{E}^r)^{\circ n}(Q^\omega_r)\right), \tag{4}$$

where $Q^\alpha_\ell = \alpha\alpha^T$ and $Q^\omega_r = \omega\omega^T$ are rank-1 matrices constituting boundary conditions for the normalization term. We use $(\mathcal{E}^r)^{\circ n}$ to denote the composition of $\mathcal{E}^r$ with itself n times, and define $(\mathcal{E}^r)^{\circ 0}$ to be the identity map acting on square matrices. For an MPS of bond dimension D over an alphabet of size d, a single transfer operator application requires time $\mathcal{O}(dD^3)$, giving a sequential runtime of $\mathcal{O}(ndD^3)$ for computing $\mathcal{Z}_n$. By representing transfer operators as $D^2 \times D^2$ matrices, this computation can be parallelized in a similar manner as described in Section 3, but at the price of increasing the total computational cost to $\mathcal{O}(nD^6)$.

4 Regular Expressions and u-MPS

While transfer operators as defined above are standard in quantum many-body physics, we now show how this transfer operator calculus can be richly generalized in the setting of sequential data. We work with regular expressions (regex) R over an alphabet Σ of size d, which can be recursively defined in terms of: (a) Single characters $c \in \Sigma$, (b) Concatenations of regex $R = R_1R_2$, (c) Unions of regex $R = R_1|R_2$, and (d) Kleene closures of regex $R = S^*$. We use Σ to denote the regex which matches a single character, and Σ^n to denote the concatenation of Σ with itself n times.

Any regex R defines a set $\mathrm{Lang}(R) \subset \Sigma^*$, the language of strings matching the pattern specified by R. While $\mathrm{Lang}(R)$ is uniquely determined by R, it is typically possible to choose multiple regex which define the same language. We assume in the following that we have chosen an unambiguous regex R, so that each string $s \in \mathrm{Lang}(R)$ matches R exactly once. This involves no loss of generality, since any ambiguous regex can be replaced by an unambiguous regex defining the same language [5]. In such cases, we will use R to also represent the subset $\mathrm{Lang}(R)$.

[2] In general, CP maps are linear operators $\mathcal{F}$ acting on square matrices by the rule $\mathcal{F}(Q) = \sum_{i=1}^K A_i Q A_i^T$. CP maps are guaranteed to send positive semidefinite (PSD) to other PSD matrices, allowing us to assume in the following that all Q_ℓ and Q_r are PSD.

Algorithm 1 Regex sampling algorithm for u-MPS

function SAMPLE(R, Q_ℓ, Q_r)
  if $R = c$ **then** *// Sample a character $c \in \Sigma$*
    return c
  else if $R = R_1 R_2$ **then** *// Sample a sequence of expressions*
    $s_1 = \text{SAMPLE}(R_1, Q_\ell, \mathcal{E}^r_{R_2}(Q_r))$
    $s_2 = \text{SAMPLE}(R_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)$
    return $s_1 s_2$
  else if $R = R_1 | R_2$ **then** *// Sample a union of expressions*
    Sample random $i \in \{1, 2\}$, with probabilities $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r)$ / $\mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)$
    $s_i = \text{SAMPLE}(e_i, Q_\ell, Q_r)$
    return s_i
  else if $R = S^*$ **then** *// Sample regex S zero or more times*
    Sample random $i \in \{\text{HALT}, \text{GO}\}$, with probabilities
      $p(\text{HALT}) = \text{Tr}(Q_\ell Q_r)/\mathcal{Z}_{S^*}(Q_\ell, Q_r)$ and $p(\text{GO}) = 1 - p(\text{HALT})$
    if $i = \text{HALT}$ **then** *// Return empty string*
      return ε
    else *// Sample one or more chars*
      return SAMPLE(SS^*, Q_ℓ, Q_r)

To each regex R, we associate a pair of generalized transfer operators $\mathcal{E}^r_R$ and $\mathcal{E}^\ell_R$, formed by summing over all strings in the language R, whose action on matrices is

$$\mathcal{E}^r_R(Q_r) = \sum_{s \in R} \mathcal{A}(s) Q_r \mathcal{A}(s)^T, \qquad \mathcal{E}^\ell_R(Q_\ell) = \sum_{s \in R} \mathcal{A}(s)^T Q_\ell \mathcal{A}(s). \tag{5}$$

While the naive sum appearing in (5) can have infinitely many terms, the action of such CP maps can still be efficiently and exactly computed in terms of the recursive definition of the regex itself. Table 1 gives the correspondence between the four primitive regex operations introduced above and the corresponding operations on CP maps. Proof of the consistency between these recursive operations and (5) for unambiguous regex is given in Appendix A.

The Kleene closure $\mathcal{E}^r_S$ in Table 1 involves an infinite summation, which is guaranteed to converge whenever the spectral norm of $\mathcal{E}^r_S$ is bounded as $\rho(\mathcal{E}^r_S) < 1$. In this case, Q^*_r can be approximated using a finite number of summands, or alternately computed exactly as the solution to the linear equation $(I - \mathcal{E}^r_S)Q^*_r = Q_r$ (see [3]).

Among other things, transfer operators can be interpreted as normalization functions for u-MPS sampling distributions. By defining $\mathcal{Z}_R(Q_\ell, Q_r) := \text{Tr}(Q_\ell \mathcal{E}^r_R(Q_r))$, we see that the normalization functions $\mathcal{Z}_n$ and $\mathcal{Z}_*$ defined above are special cases of this prescription, with boundary matrices $Q_\ell = \alpha\alpha^T, Q_r = \omega\omega^T$ and respective regex $R = \Sigma^n$ and $R = \Sigma^*$. When incorporated in a task-specific loss function (e.g. negative log likelihood), the implementation of $\mathcal{Z}_R$ in an automatic differentiation library allows this quantity to yield gradients with respect to the model parameters $\mathcal{A}$, α, and ω.

5 Sampling

The exact correspondence developed above between syntactic operations on regex and linear-algebraic operations on CP maps endows u-MPS models with rich sampling capabilities unseen in typical generative models. In particular, the function SAMPLE defined recursively in Algorithm 1 gives a means of converting any regex R into an efficient sampling procedure, whose random outputs are (for unambiguous R) unbiased samples from the conditional u-MPS distribution associated with the subset $R \subset \Sigma^*$. This is formalized in

Theorem 1. *Consider a u-MPS model with core tensor $\mathcal{A}$ and boundary vectors α and ω, along with an unambiguous regex R whose right transfer operator $\mathcal{E}^r_R$ converges. Let P_* indicate the probability distribution over arbitrary strings defined by the u-MPS, so that $\Sigma_{s \in \Sigma^*} P_*(s) = 1$. Then calling* SAMPLE($R, \alpha\alpha^T, \omega\omega^T$) *generates a random string $s \in \Sigma^*$ from the conditional u-MPS distribution $P_*(s | s \in R) = P_*(s)/P_*(R)$, where $P_*(R) := \sum_{s' \in R} P_*(s')$.*

We prove Theorem 1 in Appendix B, which also discusses sampling with ambiguous regex R. For this latter case, Algorithm 1 works identically, but returns samples from a distribution where strings s are weighted based on the number of times s matches R.

Although Algorithm 1 is written in a recursive manner, it is useful to consider the simple example $R = \Sigma^n$, a concatenation of the single-character regex Σ with itself n times, to understand the overall control flow. In this case, Algorithm 1 first attempts to sample the initial character in the string via a recursive call to $\text{SAMPLE}(\Sigma, \alpha\alpha^T, \mathcal{E}^r_{\Sigma^{n-1}}(\omega\omega^T))$. This requires $n-1$ applications of the transfer operator $\mathcal{E}^r$ to the initial right boundary matrix, and yields one new character before continuing to the right and repeating this process again.

As is common with recursive algorithms, caching intermediate information permits the naive cost of $(n-1) + (n-2) + \cdots + 1 = \mathcal{O}(n^2)$ transfer operator applications to be reduced to $\mathcal{O}(n)$. This cached version is equivalent to a simple iterative algorithm, where a sequence of right boundary matrices is first generated and saved during a right-to-left sweep, before a left-to-right sweep is used to sample text and propagate conditional information using the left boundary matrices. Using this idea, we show in Appendix C that for typical regex R, Algorithm 1 can be run with average-case runtime $\mathcal{O}(LdD^3)$ and worst-case memory usage $\mathcal{O}(LD^2)$, for L the number of characters in R, d the size of Σ, and D the bond dimension of the u-MPS.

6 Experiments

To assess the performance of u-MPS in probabilistic sequence modeling and grammatical inference, we carry out experiments on several synthetic text datasets consisting of five Tomita grammars of binary strings and a context-free "Motzkin" grammar over the trinary alphabet $\Sigma_M = \{\ (\ ,\ \&\ ,\ )\ \}$ [37, 1]. The latter consists of all strings whose parentheses are properly balanced, with no constraints placed on the $\&$ characters.

In each case we train the u-MPS on strings of a restricted length from the grammar and then sample new strings of unseen lengths from the trained u-MPS, with the model assessed on the percentage of sampled strings which match the grammar. The sampling comes in two forms, either fixed length-n sampling (corresponding to $R = \Sigma^n$), or character completion sampling, where a single character in a reference string is masked and the prefix and suffix p and s are used to guess it (corresponding to $R = p\Sigma s$). While more general sampling experiments can easily be imagined, we have chosen these tasks because they allow for direct comparisons with unidirectional and bidirectional LSTM baselines.

While unbiased fixed-length sampling is easy for u-MPS via Algorithm 1, we found that the unidirectional LSTM baseline required an additional positional encoding in its inputs to avoid rapid degeneration in the output text when sampling past the longest length seen in training. At sampling time, we vary the length scale associated with this encoding based on the desired sampling length, so that the final step of sampling is always associated with the same positional encoding vector.

We train the u-MPS and LSTM using gradient descent on a negative log likelihood (NLL) loss with the Adam [20] optimizer. For each experiment we use models of $D = 20$ and $D = 50$ hidden units in five independent trials each, with the final validation loss used to select the best model for generating samples. We use a piecewise constant learning rate between 10^{-2} and 10^{-4}, and early stopping to choose the end of training.

In the Tomita experiments (Table 2), we see u-MPS giving impressive performance, in many cases achieving perfect accuracy in sampling strings of unseen sizes within the language. This is true not only in the simpler grammars Tomita 3 and 4, but also in the more difficult Tomita 5, where valid strings satisfy the nonlocal constraint of containing an even number of `0`'s and of `1`'s. Compared to the LSTM, the correctness of the u-MPS's generated text is robust against changes in the sequence length, suggesting that the model is learning the exact grammar of the language. Given the close connection between u-MPS and regular languages this positive result is not entirely unexpected, but the fact that u-MPS can learn such structure from an unlabeled dataset without any further input is surprising.

Similar results are seen with the context-free Motzkin language (Table 3), where a fixed-length sampling task similar to the Tomita experiment is paired with a character completion task. We must use two separate baselines in this case, since each task requires a different type of RNN architecture

Table 2: Experiments on Tomita grammars 3-7 (see Appendix D for the definitions of these grammars), where all strings in the training data have lengths between 1 and 15. The trained models are used to sample strings of lengths 16 and 30, with the percentage of grammatically correct samples reported. The u-MPS consistently gives better generalization across different lengths, except for Tomita 6 which neither model is able to learn. Most of the Tomita grammars are too small to train with more than 1,000 strings, but Tomita 5 and 6 permit experiments with larger datasets.

TOMITA # (N_{train})	SAMP. LEN. 16		SAMP. LEN. 30	
	U-MPS	LSTM	U-MPS	LSTM
3 (1K)	**100.0**	90.2	**100.0**	85.6
4 (1K)	**99.9**	85.4	**99.5**	64.7
5 (1K)	**50.5**	49.0	49.1	**50.2**
5 (10K)	**100.0**	49.9	**99.9**	52.8
6 (1K)	32.1	**33.1**	33.9	**34.2**
6 (10K)	**35.9**	33.1	33.1	**34.4**
7 (1K)	**99.3**	89.2	**89.4**	29.1

Table 3: Experiments on the context-free Motzkin grammar, where the training set is fixed to contain only strings of length 15. We explore both fixed-length sampling (Samp) and character completion (Comp) tasks, where the model either samples a string from scratch, or predicts a missing character in a reference string given access to the character's prefix and suffix. In each case, the same trained u-MPS is used to give both sampling and character completion data. The bidirectional LSTM outperforms the u-MPS on shorter strings in the character completion task, but quickly degrades in accuracy as the length of the reference strings are increased.

TASK (N_{train})	SAMP. LEN. 1		SAMP. LEN. 16		SAMP. LEN. 50	
	U-MPS	LSTM	U-MPS	LSTM	U-MPS	LSTM
SAMP (1K)	**89.4**	41.7	**74.4**	41.2	**32.5**	0.0
COMP (1K)	89.4	**99.9**	69.6	**99.5**	58.8	**61.3**
SAMP (10K)	**99.3**	35.7	**99.8**	60.4	**91.6**	5.4
COMP (10K)	99.3	**100.0**	99.8	**100.0**	**92.4**	69.1

(unidirectional or bidirectional) to perform the sampling. By contrast, a trained u-MPS model can be employed in both of these settings without any task-specific adaptation, as well as in more general sentence completion tasks involving connected or disjoint regions of missing text (tasks which cannot be easily handled by common RNN models). The u-MPS does substantially better in reproducing the structure of Motzkin strings than the unidirectional LSTM, and outperforms the bidirectional LSTM when predictions are required for longer strings.

7 Conclusion

We utilize a u-MPS model for probabilistic modeling of sequence data, which we show is endowed with both significant parallelism and rich generative capabilities. Our sampling algorithm relies on a close connection between regular languages and generalized transfer operators of u-MPS, a connection we expect to extend nicely to other pairs of language classes and tensor network models. Of particular interest are tree tensor networks utilizing weight-sharing, which should be similarly capable of sampling from conditional distributions associated with context-free languages. Given the greater relevance of context-free grammars for natural language processing, we expect this direction to hold the promise of producing novel language models which can seamlessly integrate domain knowledge from linguistics to more efficiently learn and reproduce the structure of natural language.

A natural next step is scaling up u-MPS for real-world sequence modeling tasks, notably language modeling. Some current obstacles to this process are (a) the $\mathcal{O}(D^3)$ cost of certain u-MPS operations (notably, parallel evaluation and normalization functions $\mathcal{Z}_R$), and (b) the absence of well-established best-practices for training large tensor networks with gradient descent. We expect these issues to be circumvented by further directed research into the practical application of tensor networks in machine learning, a process which can be accelerated by the large number of powerful tensor network methods which have already been developed in the many-body physics community. Considering the

unexpected benefits of u-MPS for parallelism and structured text generation we have demonstrated here, we expect recurrent tensor network architectures to have a bright future in machine learning.

Broader Impact

The u-MPS model utilizes a qualitatively different means of calculating probabilities and generating samples compared to current models, which we anticipate could alter the conventional wisdom regarding how probabilistic models are trained, regularized, and applied in real-world settings. While we have so far focused mostly on sampling, the ability to compute exact probabilities of all strings matching a target regular expression could be even more important for downstream applications. A trained u-MPS model can be thought of as enabling a type of "closed-form search", where a query (literal string or regex) is evaluated to give the probability of that query appearing somewhere in a sample drawn from the u-MPS distribution. Because these probabilities are obtained as a differentiable function of the model parameters, any query or collection of queries can be immediately used to build task-specific regularizers for the model.

As a positive example of this process, u-MPS language models could be trained to directly mitigate gender bias in its generated text. Given two sets of equivalent but differently gendered phrases (for example, "his career" vs "her career"), the probability of each set could be calculated, and the difference between the two used as a regularization loss during training. This sort of fine-grained control over language also offers new possibilities for abuse though, for example by a large company or government introducing artificial bias into a consumer-facing language model product for adverse financial or political motives.

Acknowledgements

This research is supported by the Canadian Institute for Advanced Research (CIFAR AI chair program) and the MITACS Accelerate International program.

References

[1] Rafael N Alexander, Glen Evenbly, and Israel Klich. Exact holographic tensor networks for the Motzkin spin chain. *arXiv:1806.09626*, 2018.

[2] Raphael Bailly. Quadratic weighted automata: Spectral algorithm and likelihood maximization. In *Asian Conference on Machine Learning*, pages 147–163, 2011.

[3] Borja Balle, Prakash Panangaden, and Doina Precup. Singular value automata and approximate minimization. *Mathematical Structures in Computer Science*, 86(1):1–35, 2019.

[4] Yoshua Bengio and Paolo Frasconi. An EM approach to learning sequential behavior. *Advances in Neural Information Processing Systems*, 7, 1994.

[5] Ronald Book, Shimon Even, Sheila Greibach, and Gene Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 100(2):149–153, 1971.

[6] Max Born. Quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 38(11-12):803–827, 1926.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018.

[8] Song Cheng, Lei Wang, Tao Xiang, and Pan Zhang. Tree tensor networks for generative modeling. *Physical Review B*, 99(15):155131, 2019.

[9] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *arXiv:1003.4394*, 2010.

[10] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory (CoLT)*, pages 698–728, 2016.

[11] Eric DeGiuli. Random language model. *Physical Review Letters*, 122:128301, 2019.

[12] François Denis and Yann Esposito. On rational stochastic languages. *Fundamenta Informaticae*, 86(1):41–47, 2008.

[13] Mark Fannes, Bruno Nachtergaele, and Reinhard F Werner. Finitely correlated states on quantum spin chains. *Communications in mathematical physics*, 144(3):443–490, 1992.

[14] Andrew J Ferris and Guifre Vidal. Perfect sampling with unitary tensor networks. *Physical Review B*, 85(16):165146, 2012.

[15] Jakob N Foerster, Justin Gilmer, Jascha Sohl-Dickstein, Jan Chorowski, and David Sussillo. Input switched affine networks: an rnn architecture designed for interpretability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1136–1145, 2017.

[16] Angel Gallego and Román Orús. Language design as information renormalization. *arXiv:1708.01525*, 2017.

[17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

[18] Ivan Glasser, Nicola Pancotti, and J Ignacio Cirac. Supervised learning with generalized tensor networks. *arXiv:1806.05964*, 2018.

[19] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[21] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In *Conference on Learning Theory (CoLT)*, 2018.

[22] Alex Monras, Almut Beige, and Karoline Wiesner. Hidden quantum markov models and non-adaptive read-out of many-body states. *arXiv:1002.2337*, 2010.

[23] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

[24] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[25] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines. In *International Conference on Learning Representations (ICLR)*, 2017.

[26] Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550, 2019.

[27] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems*, 2017.

[29] David Perez-García, Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Matrix product state representations. *Quantum Information and Computation*, 7(5-6):401–430, 2007.

[30] Vasily Pestun, John Terilla, and Yiannis Vlassopoulos. Language as a matrix product state. *arXiv:1711.01416*, 2017.

[31] Vasily Pestun and Yiannis Vlassopoulos. Tensor network language model. *arXiv:1710.10248*, 2017.

[32] Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting weighted automata and recurrent neural networks through spectral learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[33] Siddarth Srinivasan, Geoff Gordon, and Byron Boots. Learning hidden quantum markov models. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[34] James Stokes and John Terilla. Probabilistic modeling with matrix product states. *Entropy*, 21(12), 2019.

[35] E Miles Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, 2018.

[36] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.

[37] Masaru Tomita. Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108, 1982.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Steven R White. Density matrix formulation for quantum renormalization groups. *Physical review letters*, 69(19):2863, 1992.

REGEX R =	c	$R_1 R_2$	$R_1 \vert R_2$	S^*
$\boldsymbol{\mathcal{E}^r_R(Q_r)}$ =	$\mathcal{A}(c) Q_r \mathcal{A}(c)^T$	$\mathcal{E}^r_{R_1}(\mathcal{E}^r_{R_2}(Q_r))$	$\mathcal{E}^r_{R_1}(Q_r) + \mathcal{E}^r_{R_2}(Q_r)$	$\sum_{n=0}^{\infty} (\mathcal{E}^r_S)^{\circ n}(Q_r)$
$\boldsymbol{\mathcal{E}^\ell_R(Q_\ell)}$ =	$\mathcal{A}(c)^T Q_\ell \mathcal{A}(c)$	$\mathcal{E}^\ell_{R_2}(\mathcal{E}^\ell_{R_1}(Q_\ell))$	$\mathcal{E}^\ell_{R_1}(Q_\ell) + \mathcal{E}^\ell_{R_2}(Q_\ell)$	$\sum_{n=0}^{\infty} (\mathcal{E}^\ell_S)^{\circ n}(Q_\ell)$

A Completely Positive Maps and Generalized Transfer Operators

In this section, we give definitions and known results concerning completely positive (CP) maps and regular expressions (regex), as well as further details regarding the assignment of generalized transfer operators to regex[3]. We conclude with a proof that the recursive definition of generalized transfer operators given in Table 1 (repeated here for convenience) has an equivalent representation in terms of a weighted sum over all strings in the regex, which for unambiguous regex gives precisely the simple form of (5).

We say that a matrix $Q \in \mathbb{R}^{D \times D}$ is positive semidefinite (PSD) if it is (a) Symmetric ($Q^T = Q$), and (b) satisfies $v^T Q v \geq 0$ for every $v \in \mathbb{R}^D$. If Q further satisfies the property that $v^T Q v = 0$ only when $v = 0$, then we call it positive definite. Given a PSD matrix Q, the diagonal elements of Q will necessarily be non-negative. For any vector v, the rank-1 matrix vv^T is necessarily a PSD matrix, and all PSD matrices $Q \in \mathbb{R}^{D \times D}$ can be expressed as the weighted sum of (at most) D such rank-1 matrices. This can be used to show that for any PSD matrices Q and Q', $\mathrm{Tr}(QQ') \geq 0$.

It is common in quantum mechanics to regard PSD matrices as a generalized form of probabilistic states [23], a viewpoint which allows us to consider the matrices Q_ℓ, Q_r as probabilistic latent states of a u-MPS. To this end, the family of completely positive (CP) maps is the natural generalization of stochastic maps, which act on these PSD matrices. A map $\mathcal{E}$ sending PSD $Q \in \mathbb{R}^{D \times D}$ to $Q' = \mathcal{E}(Q) \in \mathbb{R}^{D' \times D'}$ is said to be CP if it admits a Kraus representation, consisting of $r \geq 1$ matrices $A_i \in \mathbb{R}^{D' \times D}$ such that $\mathcal{E}$ can be expressed as:

$$\mathcal{E}(Q) = \sum_{i=1}^{r} A_i Q A_i^T \tag{6}$$

The condition (6) implies in particular that $\mathcal{E}(Q)$ is PSD if Q is. The matrices in (6) are called the Kraus operators of the map, and the same CP map $\mathcal{E}$ can be given multiple Kraus representations with inequivalent values of r. The minimum value of r such that $\mathcal{E}$ can be represented as (6) is called the rank of $\mathcal{E}$, and is always bounded as $r \leq D^2$. Nonetheless, Kraus representations with a greater number of Kraus operators can be useful for understanding the action of the map, as we will see below.

By taking the transpose of all the Kraus operators appearing in (6), we obtain a new CP map $\mathcal{E}^T$, which is the adjoint of $\mathcal{E}$. Mathematically, this means that for any CP map $\mathcal{E}$ and positive matrices Q_ℓ, Q_r, the equality $\mathrm{Tr}(Q_\ell \mathcal{E}(Q_r)) = \mathrm{Tr}(\mathcal{E}^T(Q_\ell) Q_r)$. For greater clarity, in the context of sequence modeling with u-MPS we frequently refer to a CP map and its adjoint as "right" and "left" maps $\mathcal{E}^r$ and $\mathcal{E}^\ell$, rather than $\mathcal{E}$ and $\mathcal{E}^T$.

The term "transfer operator" is common in many-body physics, and in our setting refers to a CP map $\mathcal{E}$ obtained from the Kraus representation with $A_i = \mathcal{A}(\varphi^{-1}(i))$, for $\mathcal{A}$ a u-MPS core tensor and $\varphi : \Sigma \to [d]$ a bijection mapping characters c in the size-d alphabet Σ to the numbers $\{1, 2, \ldots, d\}$ (see Figure 1f). In Section 4 we introduced a generalization of this standard notion of transfer operator to include a number of other CP maps $\mathcal{E}_R$ associated with an arbitrary regular expression R, whose recursive definition is given in Table 1. Using Σ to also denote the regex matching any single character in our alphabet, the standard transfer operator emerges as the special case $R = \Sigma$.

We sometimes assume that our regex R is unambiguous, in the sense that any string s matching R matches in exactly one way. This assumption can be made without loss of generality, since any ambiguous regex R can be converted into an unambiguous regex R' accepting the same set of strings [5]. For example, the ambiguous regex $R = a^* a^*$ matches the string $s = a$ in two ways, but can be replaced with the equivalent unambiguous regex $R' = a^*$.

[3] We present all definitions and results in this section in terms of real-valued matrices, but the corresponding statements for complex-valued matrices and CP maps is obtained by replacing matrix transposes Q^T by Hermitian adjoints $Q^\dagger$, representing the transposed and complex-conjugated counterpart of Q.

Any regex R can be built inductively from (a) Single characters $c \in \Sigma$, (b) Concatenations of regex $R = R_1 R_2$, (c) Unions of regex $R = R_1|R_2$, and (d) Kleene closures of regex $R = S^*$. We prove by induction over the structure of R that any generalized right transfer operator $\mathcal{E}_R^r$ defined by the recursive procedure in Table 1 acts according to a generalization of (5), as stated in

Theorem 2. *Consider the generalized transfer operators $\mathcal{E}_R^r$ and $\mathcal{E}_R^\ell$ associated with an arbitrary regex R and a u-MPS with core tensor $\mathcal{A}$, which are defined by the recursive rules in Table 1. Then $\mathcal{E}_R^r$ converges if and only if $\mathcal{E}_L^r$ converges, and in this case the transfer operators are described by the Kraus representations,*

$$\mathcal{E}_R^r(Q_r) = \sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s) Q_r \mathcal{A}(s)^T, \qquad \mathcal{E}_R^\ell(Q_\ell) = \sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s)^T Q_\ell \mathcal{A}(s), \tag{7}$$

where $|s|_R$ denotes the number of times the string s matches the regex R. For unambiguous regex, this Kraus representation is identical to that of (5)*.*

Proof. For each of the four types of regex R, we make the inductive assumption that the subexpressions of R (if any) satisfy (7), and use this to prove that the transfer operator $\mathcal{E}_R^r$ satisfies (7). This allows us to immediately prove the corresponding statement for the left transfer operator $\mathcal{E}_R^\ell$.

$\mathbf{R = c}$: Apparent from Table 1 and the single string which matches R, $s = c$.

$\mathbf{R = R_1 R_2}$: Assume R_1 and R_2 both satisfy (7). Table 1 gives:

$$\begin{aligned}\mathcal{E}_{R_1R_2}^r(Q_r) = \mathcal{E}_{R_1}^r(\mathcal{E}_{R_2}^r(Q_r)) &= \sum_{s_1\in\Sigma^*}\sum_{s_2\in\Sigma^*} |s_1|_{R_1}|s_2|_{R_2}\mathcal{A}(s_1)\mathcal{A}(s_2) Q_r \mathcal{A}(s_2)^T\mathcal{A}(s_1)^T \\ &= \sum_{s_1\in\Sigma^*}\sum_{s_2\in\Sigma^*} |s_1|_{R_1}|s_2|_{R_2}\mathcal{A}(s_1s_2) Q_r \mathcal{A}(s_1s_2)^T = \sum_{s\in\Sigma^*} |s|_{R_1R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the second-to-last equality we used the compositional property $\mathcal{A}(s_1)\mathcal{A}(s_2) = \mathcal{A}(s)$, while in the last equality we used the identity $|s|_{R_1R_2} = \sum_{s_1 s_2 = s} |s_1|_{R_1}|s_2|_{R_2}$, where the sum over s_1, s_2 represents all possible partitions of s into a prefix and suffix.

$\mathbf{R = R_1|R_2}$: Assume R_1 and R_2 both satisfy (7). Table 1 gives:

$$\begin{aligned}\mathcal{E}_{R_1|R_2}^r(Q_r) = \mathcal{E}_{R_1}^r(Q_r) + \mathcal{E}_{R_2}^r(Q_r) &= \left(\sum_{s\in\Sigma^*} |s|_{R_1}\mathcal{A}(s) Q_r \mathcal{A}(s)^T\right) + \left(\sum_{s\in\Sigma^*} |s|_{R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T\right) \\ &= \sum_{s\in\Sigma^*} |s|_{R_1|R_2}\mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the final equality we have used the identity $|s|_{R_1|R_2} = |s|_{R_1} + |s|_{R_2}$.

$\mathbf{R = S^*}$: Assume S satisfies (7). The operator $\mathcal{E}_{S^*}^r$ will converge only when the spectral norm of $\mathcal{E}_S^r$ satisfies $\rho(\mathcal{E}_S^r) < 1$, in which case Table 1 gives:

$$\begin{aligned}\mathcal{E}_{S^*}^r(Q_r) = \sum_{n=0}^{\infty} (\mathcal{E}_S^r)^{\circ n}(Q_r) &= \sum_{n=0}^{\infty}\sum_{s_1\cdots s_n\in\Sigma^*} |s_1|_S \cdots |s_n|_S \, \mathcal{A}(s_1\cdots s_n) Q_r \mathcal{A}(s_1\cdots s_n)^T \\ &= \sum_{s\in\Sigma^*} |s|_{S^*} \, \mathcal{A}(s) Q_r \mathcal{A}(s)^T.\end{aligned}$$

In the final equality we have used the identity $|s|_{S^*} = \sum_{n=0}^{\infty}\sum_{s_1\cdots s_n = s} |s_1|_S \cdots |s_n|_S$, where the sum over $s_1, \ldots, s_n$ represents all possible partitions of s into n contiguous pieces.

While we only characterized the action of the right transfer operators $\mathcal{E}_R^r$, substituting all matrices $\mathcal{A}(s)$ with their transposed counterparts $\mathcal{A}(s)^T$ immediately yields the corresponding characterization for the action of $\mathcal{E}_R^\ell$. In this latter case, the direction-reversing identity $\mathcal{A}(s_1s_2)^T = \mathcal{A}(s_2)^T\mathcal{A}(s_1)^T$ is accounted for by the transfer operator correspondence $\mathcal{E}_{R_1R_2}^\ell = \mathcal{E}_{R_2}^\ell \mathcal{E}_{R_1}^\ell$.

Because $\mathcal{E}_R^r$ and $\mathcal{E}_R^\ell$ are adjoints of each other, their eigenvalue spectra are identical, and therefore $\mathcal{E}_R^r$ converges if and only if $\mathcal{E}_R^\ell$ converges. Finally for unambiguous regex R, the quantity $|s|_R \in \{0, 1\}$, giving the equality $\sum_{s\in\Sigma^*} |s|_R \mathcal{A}(s) Q_r \mathcal{A}(s)^T = \sum_{s\in R} \mathcal{A}(s) Q_r \mathcal{A}(s)^T$ which proves (5).

□

Although it is not obvious a priori when a transfer operator $\mathcal{E}_R^r$ will converge for a given regex R and core tensor $\mathcal{A}$, it is clear that the Kleene closure is the only operation permitting divergence. Consequently, a regex R will converge only when all of its subexpressions of the form S_i^* have spectral norm $\rho(\mathcal{E}_{S_i}^r) < 1$. Note that any S^* for which S accepts the empty string is guaranteed to produce a divergent transfer operator $\mathcal{E}_{S^*}^r$, so that in particular any transfer operator of the form $\mathcal{E}_{(S^*)^*}^r$ is divergent.

B Proof of Theorem 1

In order to prove Theorem 1, we first prove a more general Lemma 1, which characterizes the probability distribution $P_R(s, Q_\ell, Q_r)$ of strings output by $\text{SAMPLE}(R, Q_\ell, Q_r)$ for arbitrary R.

Lemma 1. *Consider a u-MPS model with core tensor $\mathcal{A}$ and a regex R for which the generalized right transfer operator $\mathcal{E}_R^r$ defined recursively by Table 1 converges. Then for any PSD matrices Q_ℓ, Q_r, the probability distribution of strings output by $\text{SAMPLE}(R, Q_\ell, Q_r)$ is $P_R(s, Q_\ell, Q_r) = |s|_R \tilde{P}(s, Q_\ell, Q_r)/\mathcal{Z}_R(Q_\ell, Q_r)$, where $\tilde{P}(s, Q_\ell, Q_r) = \text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))$, $\mathcal{Z}_R(Q_\ell, Q_r) = \text{Tr}(Q_\ell \mathcal{E}_R^r(Q_r))$, and $|s|_R$ counts the number of times the string s matches the regex R.*

Proof. We prove Lemma 1 by induction over the structure of R, where we assume that sampling from a regex subexpression R' of R with any boundary matrices Q'_ℓ and Q'_r produces strings from the distribution $P_{R'}(s, Q'_\ell, Q'_r)$. For each of the four cases of regex formation, we use this inductive assumption to show that sampling from R produces strings from the distribution $P_R(s, Q_\ell, Q_r)$.

$\mathbf{R = c}$: From Algorithm 1, $\text{SAMPLE}(c, Q_\ell, Q_r)$ will always output the string $s = c$. Because the quantity $|s|_c$ is 1 when $s = c$ and 0 otherwise, the sampling distribution can be written as

$$P_c(s, Q_\ell, Q_r) = |s|_c = |s|_c \frac{\text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_c^r(Q_r))} = |s|_c \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_c(Q_\ell, Q_r)}.$$

$\mathbf{R = R_1 R_2}$: From Algorithm 1, $\text{SAMPLE}(R_1 R_2, Q_\ell, Q_r)$ will first output a string s_1 from $\text{SAMPLE}(R_1, Q_\ell, \mathcal{E}_{R_2}^r(Q_r))$, then use s_1 to output a string s_2 from $\text{SAMPLE}(R_2, \mathcal{E}_{s_1}^\ell(Q_\ell), Q_r)$. Using our inductive assumption for R_1 and R_2, the probability assigned to the output string s from all possible partitions into a prefix and suffix as $s_1 s_2 = s$ is

$$\begin{aligned}
P_{R_1 R_2}(s, Q_\ell, Q_r) &= \sum_{s_1 s_2 = s} P_{R_1}(s_1, Q_\ell, \mathcal{E}_{R_2}^r(Q_r)) \cdot P_{R_2}(s_2, \mathcal{E}_{s_1}^\ell(Q_\ell), Q_r) \\
&= \sum_{s_1 s_2 = s} \left(|s_1|_{R_1} \frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1}^r(\mathcal{E}_{R_2}^r(Q_r)))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1}^r(\mathcal{E}_{R_2}^r(Q_r)))} \right) \left(|s_2|_{R_2} \frac{\text{Tr}(\mathcal{E}_{s_1}^\ell(Q_\ell) \mathcal{E}_{s_2}^r(Q_r))}{\text{Tr}(\mathcal{E}_{s_1}^\ell(Q_\ell) \mathcal{E}_{R_2}^r(Q_r))} \right) \\
&= \sum_{s_1 s_2 = s} |s_1|_{R_1} |s_2|_{R_2} \left(\frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1 R_2}^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1 R_2}^r(Q_r))} \right) \left(\frac{\text{Tr}(Q_\ell \mathcal{E}_{s_1 s_2}^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{s_1 R_2}^r(Q_r))} \right) \\
&= |s|_{R_1 R_2} \frac{\text{Tr}(Q_\ell \mathcal{E}_s^r(Q_r))}{\text{Tr}(Q_\ell \mathcal{E}_{R_1 R_2}^r(Q_r))} = |s|_{R_1 R_2} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_1 R_2}(Q_\ell, Q_r)}.
\end{aligned}$$

In the third equality above, we use the composition rule $\mathcal{E}_{s'}^r \mathcal{E}_{R'}^r = \mathcal{E}_{s'R'}^r$ and the adjunction rule $\text{Tr}(\mathcal{E}_{s'}^\ell(Q'_\ell) Q'_r) = \text{Tr}(Q'_\ell \mathcal{E}_{s'}^r(Q'_r))$, while in the fourth equality, we use the identity $|s|_{R_1 R_2} = \sum_{s_1 s_2 = s} |s_1|_{R_1} |s_2|_{R_2}$.

$\mathbf{R} = \mathbf{R_1}|\mathbf{R_2}$: From Algorithm 1, SAMPLE$(R_1|R_2, Q_\ell, Q_r)$ will first pick a random index $i \in 1, 2$ with probability $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r) \;/\; \mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)$, and then use this to output a string s from SAMPLE(R_i, Q_ℓ, Q_r). Using our inductive assumption, the probability assigned to the output string s is

$$\begin{aligned} P_{R_1|R_2}(s, Q_\ell, Q_r) &= \sum_{i\in\{1,2\}} p(i) \cdot P_{R_i}(s_i, Q_\ell, Q_r) \\ &= \sum_{i\in\{1,2\}} \left(\frac{\mathcal{Z}_{R_i}(Q_\ell, Q_r)}{\mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)}\right)\left(|s|_{R_i}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_i}(Q_\ell, Q_r)}\right) \\ &= |s|_{R_1|R_2}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{R_1|R_2}(Q_\ell, Q_r)}. \end{aligned}$$

In the final equality, we have used the identity $|s|_{R_1|R_2} = |s|_{R_1} + |s|_{R_2}$.

$\mathbf{R} = \mathbf{S}^*$: To sample from the regex R we must have the infinite sum defining $\mathcal{E}^r_R$ in Table 1 converge, which is guaranteed by the assumptions of Lemma 1. Given this convergence, calling SAMPLE(S^*, Q_ℓ, Q_r) will either output the empty string $s = \varepsilon$, or else call SAMPLE(SS^*, Q_ℓ, Q_r). In the latter case, the concatenation rule will then sample some $s' \in S$ before calling SAMPLE$(S^*, \mathcal{E}^\ell_{s'}(Q_\ell), Q_r)$ to sample some random number $n \geq 0$ occurrences of S.

We denote the unnormalized collection of probabilities associated with strings produced from exactly n occurrences of S as $P^{(n)}_{S^*}$, and we use an inductive proof to show that $P^{(n)}_{S^*} = p(n)P_{S^n}$, for $p(n, Q_\ell, Q_r) = \mathcal{Z}_{S^n}(Q_\ell, Q_r)/\mathcal{Z}_{S^*}(Q_\ell, Q_r)$. In other words, our recursive sampling procedure for S^* is equivalent to first sampling a random length using $p(n)$, then calling the corresponding SAMPLE(S^n, Q_ℓ, Q_r).

Base case $n = 0$**:** The regex S^0 matches only the empty string $s = \varepsilon$, and from Algorithm 1, this occurs with probability

$$P^{(0)}_{S^*}(s, Q_\ell, Q_r) = \frac{\mathrm{Tr}(Q_\ell Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = \frac{\mathcal{Z}_{S^0}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = p(0)P_{S^0}(s, Q_\ell, Q_r),$$

where we have used the identity $\mathcal{E}^r_{S^0} = \mathcal{E}^r_\varepsilon = I$, and the fact that $P_{S^0}(s)$ is 1 for $s = \varepsilon$ and 0 otherwise.

Step case $n+1$**:** From Algorithm 1, the probability of sampling a string $s = s_1 s_2$ with s_1 matching S and s_2 matching S^n is

$$\begin{aligned} P^{(n+1)}_{S^*}(s, Q_\ell, Q_r) &= \sum_{s_1 s_2 = s}\left(1 - \frac{\mathrm{Tr}(Q_\ell Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\right) P_S(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r)) P^{(n)}_{S^*}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) \\ &= \sum_{s_1 s_2 = s}\left(\frac{\mathcal{Z}_{SS^*}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\right)\left(|s_1|_S\frac{\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}{\mathcal{Z}_S(Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}\right) P^{(n)}_{S^*}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) \\ &= \sum_{s_1 s_2 = s} |s_1|_S\frac{\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r))}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}\frac{\mathcal{Z}_{S^n}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^*}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}|s_2|_{S^n}\frac{\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^n}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)} \\ &= \sum_{s_1 s_2 = s} |s_1|_S|s_2|_{S^n}\frac{\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} = \sum_{s_1 s_2 = s} |s_1|_S|s_2|_{S^n}\frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} \\ &= \frac{\mathcal{Z}_{S^{n+1}}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}|s|_{S^{n+1}}\frac{\tilde{P}(s_1 s_2, Q_\ell, Q_r)}{\mathcal{Z}_{S^{n+1}}(Q_\ell, Q_r)} = p(n+1, Q_\ell, Q_r)P_{S^{n+1}}(s, Q_\ell, Q_r). \end{aligned}$$

In the above we used the following identities: $\mathcal{Z}_{S^*}(Q_\ell, Q_r) = \mathrm{Tr}(Q_\ell Q_r) + \mathcal{Z}_{SS^*}(Q_\ell, Q_r)$ (second equality), $\mathcal{Z}_S(Q_\ell, \mathcal{E}^r_{S^*}(Q_r)) = \mathcal{Z}_{SS^*}(Q_\ell, Q_r)$ (third equality), $\tilde{P}(s_1, Q_\ell, \mathcal{E}^r_{S^*}(Q_r)) = \mathcal{Z}_{S^*}(\mathcal{E}^\ell_{s_1}(Q_\ell), Q_r)$ (fourth equality), $\tilde{P}(s_2, \mathcal{E}^\ell_{s_1}(Q_\ell), Q_r) = \tilde{P}(s_1 s_2, Q_\ell, Q_r)$ (fifth equality), and $|s|_{S^{n+1}} = \sum_{s_1 s_2 = s} |s_1|_S|s_2|_{S^n}$ (sixth equality).

With this inductive characterization of the unnormalized distributions $P^{(n)}_{S^*}$, we can finally show

$$\begin{aligned} P_{S^*}(s, Q_\ell, Q_r) &= \sum_{n=0}^{\infty} P^{(n)}_{S^*}(s, Q_\ell, Q_r) = \sum_{n=0}^{\infty} \frac{\mathcal{Z}_{S^n}(Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)} |s|_{S^n} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^n}(Q_\ell, Q_r)} \\ &= |s|_{S^*} \frac{\tilde{P}(s, Q_\ell, Q_r)}{\mathcal{Z}_{S^*}(Q_\ell, Q_r)}, \end{aligned}$$

where we have used the identity $\sum_{n=0}^{\infty} |s|_{S^n} = |s|_{S^*}$ in the last equality.

□

Having proved Lemma 1, we can now prove Theorem 1 as a simple corollary, which is restated here for ease of reference.

Theorem. *Consider a u-MPS model with core tensor $\mathcal{A}$ and boundary vectors α and ω, along with an unambiguous regex R whose right transfer operator $\mathcal{E}^r_R$ converges. Let P_* indicate the probability distribution over arbitrary strings defined by the u-MPS, so that $\Sigma_{s\in\Sigma^*} P_*(s) = 1$. Then calling* $\text{SAMPLE}(R, \alpha\alpha^T, \omega\omega^T)$ *generates a random string $s \in \Sigma^*$ from the conditional u-MPS distribution $P_*(s|s \in R) = P_*(s)/P_*(R)$, where $P_*(R) := \sum_{s'\in R} P_*(s')$.*

Proof. From Lemma 1, we know the probability distribution of strings output by $\text{SAMPLE}(R, \alpha\alpha^T, \omega\omega^T)$, as well as $P_*(s) = \text{SAMPLE}(\Sigma^*, \alpha\alpha^T, \omega\omega^T)$. This characterization lets us show

$$\begin{aligned} P_R(s, \alpha\alpha^T, \omega\omega^T) &= |s|_R \frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\mathcal{Z}_R(\alpha\alpha^T, \omega\omega^T)} = |s|_R \frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\text{Tr}(\alpha\alpha^T \mathcal{E}^r_R(\omega\omega^T))} \\ &= |s|_R \left(\frac{\tilde{P}(s, \alpha\alpha^T, \omega\omega^T)}{\mathcal{Z}_{\Sigma^*}(\alpha\alpha^T, \omega\omega^T)} \right) \left(\frac{\mathcal{Z}_{\Sigma^*}(\alpha\alpha^T, \omega\omega^T)}{\sum_{s'\in R} \tilde{P}(s', \alpha\alpha^T, \omega\omega^T)} \right) \\ &= |s|_R \frac{P_{\Sigma^*}(s, \alpha\alpha^T, \omega\omega^T)}{\sum_{s'\in R} P_{\Sigma^*}(s', \alpha\alpha^T, \omega\omega^T)} = \begin{cases} P_*(s)/P_*(R), & \text{if } s \in R \\ 0, & \text{otherwise} \end{cases} \\ &= P_*(s|s \in R). \end{aligned}$$

In the third equality we used (7) in Theorem 2 (which reduces to (5) for an unambiguous R) to express $\mathcal{E}^r_R$ as $\mathcal{E}^r_R = \sum_{s\in R} \mathcal{E}^r_s$, while also introducing a normalization factor associated with Σ^* to the numerator and denominator. In the last equality we have used the definition of the conditional probability distribution associated with s matching the regex R, and have also utilized the fact that $|s|_R$ is either 1 or 0 for unambiguous regex.

□

C Runtime Analysis

Algorithm 1 is written as a recursive procedure, which makes its runtime analysis nontrivial. We show here that this sampling procedure can in most cases be carried out using compute and storage costs which scale linearly with the length L_R of the defining regex R. As a technical assumption, we require the star height of R to be bounded, where the star height $h_*(R)$ is defined recursively as $h_*(c) = 0$, $h_*(R_1R_2) = h_*(R_1|R_2) = \max(h_*(R_1), h_*(R_2))$, and $h_*(S^*) = 1 + h_*(S)$. In practice this assumption is very mild.

Theorem 3. *Consider a core tensor $\mathcal{A}$ and regex R of length L_R with bounded star height, for which the associated right transfer operator $\mathcal{E}^r_R$ converges. Then calling* $\text{SAMPLE}(R, Q_\ell, Q_r)$ *will return a random string of mean length $\langle n \rangle = \mathcal{O}(L_R)$, with average-case compute cost of $C_R = \mathcal{O}(L_R d D^3)$ and worst-case memory cost of $M_R = \mathcal{O}(L_R D^2)$.*

Proof. We again utilize a proof by induction, with the additional assumption that C_R is also an upper bound on the expected cost of applying the transfer operator $\mathcal{E}^r_R$. For the regex length L_R, we consider the characters $|$, $($, $)$, *, and c (for $c \in \Sigma$) as having length 1, along with the single-character

regex Σ. This definition of length is closer to that of real-world regex, where the metacharacter "." corresponds to our Σ.

In order to utilize caching in Algorithm 1, we replace the simple recursive case of binary regex concatenation $R = R_1 R_2$ with a maximal concatenation of smaller regex $R = R_1 R_2 \cdots R_K$, where each R_i is either a single character, a union of regex, or a Kleene closure. Such concatenations are the only place where caching is utilized, allowing us to bound the memory usage in terms of the regex length L_R, rather than the random string length n_R. We don't include the non-cache memory usage required to hold our u-MPS parameters and intermediate variables, which is in every case $\mathcal{O}(dD^2)$.

Given that the length n_R of an output sample is typically a random variable, we first show that the mean length is bounded as $\langle n_R \rangle = \mathcal{O}(L_R)$. It is apparent that for any regex R constructed without Kleene closures, we have the stronger bound $n_R \leq L_R$. We therefore start our inductive proof with this last remaining case of Kleene closures, showing that $\langle n_R \rangle = \mathcal{O}(L_R)$ for all regex R with bounded star height. We then use this result to characterize the compute and memory requirements of Algorithm 1.

$\mathbf{R} = \mathbf{S}^*$: In order for $\mathcal{E}^r_{S^*}$ to converge, we must have the spectral radius of $\mathcal{E}^r_S$ be $\lambda_S := \rho(\mathcal{E}^r_S) < 1$. Noting that this implies $\mathrm{Tr}(Q_\ell \mathcal{E}^r_S(Q_r)) \leq \lambda_S \,\mathrm{Tr}(Q_\ell Q_r)$ for any boundary matrices Q_ℓ, Q_r, we find that the probability of obtaining m occurrences of S is upper bounded as

$$p(m) = \mathrm{Tr}(Q_\ell (\mathcal{E}^r_S)^{\circ m}(Q_r))/\mathcal{Z}_R(Q_\ell, Q_r) \leq \lambda_S^m \,\mathrm{Tr}(Q_\ell Q_r)/\mathcal{Z}_R(Q_\ell, Q_r) = \lambda_S^m \, p(0).$$

Given this exponentially decaying upper bound, the output of SAMPLE(S^*) on average will consist of $\langle m \rangle = \mathcal{O}(\chi_S)$ calls to SAMPLE(S), where $\chi_S := \lambda_S^{-1}$. Assuming we can obtain a boundary-independent upper bound on the expected length of SAMPLE(S), then this proves that $\langle n_{S^*} \rangle = \mathcal{O}(\chi_S \langle n_S \rangle)$.

If S itself contains expressions with deeply nested Kleene closures then this task becomes difficult, with the above bound translating to $\langle n_R \rangle = \mathcal{O}(\chi^{h_*(R)} L_R)$, for $h_*(R)$ the star height of R and χ the maximum χ_{S_i} among all nested subexpressions $(S_i)^*$ within R. However, if we assume R has bounded star height, then this reduces to $\langle n_R \rangle = \mathcal{O}(\chi^{h_*(R)} L_R) = \mathcal{O}(L_R)$, our desired bound.

Moving on to a consideration of the resource costs of SAMPLE(S^*), we make the inductive assumption that a single call to SAMPLE(S) has average-case runtime of $\mathcal{O}(L_S d D^3)$ and worst-case memory usage of $\mathcal{O}(L_S D^2)$. Algorithm 1 in this case will m samples from S, using the same right boundary condition Q_r^* each time. This leads to regex length, runtime, and memory usage of

$$\begin{aligned} L_R &= L_S + 1 = \mathcal{O}(L_S), \qquad C_R = \mathcal{O}(\langle m \rangle C_S) = \mathcal{O}(\chi_S L_S d D^3) = \mathcal{O}(L_R d D^3), \\ M_R &= M_S = \mathcal{O}(L_S D^2) = \mathcal{O}(L_R D^2), \end{aligned}$$

where the last equality of C_R uses the bounded star height of R. We finally note that the above bound on C_R also applies to the transfer operator $\mathcal{E}^r_{S^*}$, whose action on Q_r can be approximated to arbitrary precision $\epsilon = \exp(\mathcal{O}(-m/\chi_S))$ using m applications of $\mathcal{E}^r_S$.

$\mathbf{R} = \sigma$, for $\sigma =$ c or $\mathbf{\Sigma}$: For the case of $R = c$, no resources are required for sampling. For $R = \Sigma$, the sampling procedure costs $C_\Sigma = \mathcal{O}(dD^3)$, which also gives an upper bound on the cost of applying the transfer operator $\mathcal{E}^r_\sigma$. The regex and output string lengths are both 1 here and no caching is involved, so

$$L_\sigma = 1, \qquad C_\sigma = \mathcal{O}(dD^3) = \mathcal{O}(L_\sigma d D^3), \qquad M_\sigma = 0 = \mathcal{O}(L_\sigma D^2).$$

$\mathbf{R} = \mathbf{R_1 R_2 \cdots R_K}$: When evaluating SAMPLE$(R_1 \cdots R_K, Q_\ell, Q_r)$, we first compute and cache the K right boundary matrices $Q_r^{(1)}, Q_r^{(2)}, \ldots, Q_r^{(K)}$ in a right-to-left sweep, via the rules $Q_r^{(K)} = Q_r$ and $Q_r^{(i-1)} = \mathcal{E}^r_{R_i}(Q_r^{(i)})$. This has a memory cost of $M_R = \mathcal{O}(KD^2)$. With these right boundary matrices in hand, we then use a left-to-right sweep to obtain strings $s_1, s_2, \ldots, s_K$ via repeated calls to SAMPLE$(R_i, Q_\ell^{(i)}, Q_r^{(i)})$, where the left boundary matrices are defined as $Q_\ell^{(1)} = Q_\ell$ and $Q_\ell^{(i+1)} = \mathcal{E}^\ell_{s_i}(Q_\ell^{(i)})$. No caching of the $Q_\ell^{(i)}$ is required, and each call to SAMPLE$(R_i, Q_\ell^{(i)}, Q_r^{(i)})$ generally involves some additional memory usage, which is freed immediately afterwards.

Applying our inductive assumption about the runtime and memory usage of each of the transfer operators and SAMPLE calls for the subexpressions $R_1, \ldots, R_K$, we get

$$L_R = \sum_{i=1}^{K} L_{R_i}, \qquad C_R = \mathcal{O}\left(\sum_{i=1}^{K} C_{R_i}\right) = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i} dD^3\right) = \mathcal{O}(L_R dD^3),$$
$$M_R = \mathcal{O}(KD^2) + \max_i(M_{R_i}) = \mathcal{O}(KD^2) + \mathcal{O}(\max_i(L_{R_i})D^2) = \mathcal{O}(L_R D^2).$$

$\mathbf{R} = \mathbf{R_1}|\mathbf{R_2}|\cdots|\mathbf{R_K}$: To evaluate SAMPLE$(R_1|\cdots|R_K, Q_\ell, Q_r)$, we must first sample a random i from the distribution $p(i) = \mathcal{Z}_{R_i}(Q_\ell, Q_r)/\mathcal{Z}_R(Q_\ell, Q_r)$, then call SAMPLE$(R_i, Q_\ell, Q_r)$. This gives the following characterization of the overall runtime and memory usage

$$L_R = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i}\right), \qquad C_R = \mathcal{O}\left(\sum_{i=1}^{K} C_{R_i}\right) = \mathcal{O}\left(\sum_{i=1}^{K} L_{R_i} dD^3\right) = \mathcal{O}(L_R dD^3),$$
$$M_R = \max_i(M_{R_i}) = \mathcal{O}(\max_i(L_{R_i})D^2) = \mathcal{O}(L_R D^2).$$

□

D Experimental Details

Table 4: Definition of Tomita grammars 3-7 given in [4], which states a necessary and sufficient condition for a string to belong to each grammar. Tomita grammars 1 and 2 correspond to the respective family of strings 1^n and $(01)^*$, and are unused because of their small size and simple structure.

Tomita #	3	4	5	6	7
Description	Doesn't contain $1^{2n+1}0^{2m+1}$ as a substring	Doesn't contain 000 as a substring	Contains an even number of 0's and 1's	Number of 0's minus number of 1's is a multiple of 3	Has the form $0^*1^*0^*1^*$

The u-MPS model we utilized was built from scratch in JAX [7], while the LSTM module from PyTorch [28] was used for the baseline. The LSTMs are single-layer models with 20 or 50 hidden units (20 or 50 in each direction for the bidirectional LSTM), and a linear decoder and softmax output layer used to obtain character probabilities. The bond dimension of the u-MPS was similar chosen to be 20 or 50, and for both types of models, five independent trials were used for each number of hidden states and the model with the lowest validation error was used to produce the sampling statistics reported in Section 6.

For each grammar, the models were trained on either 1,000 or 10,000 randomly chosen strings, with 1,000 strings used as a held-out validation set. The sampling percentages for Table 2 and the sampling tasks of Table 3 were obtained from sampling 1,000 random strings from the respective models, while the completion tasks of Table 3 used 1,000 random strings from a held-out reference set, where the models were used to infer each character in each string when all other characters were used as bidirectional context.

In all experiments, models were trained with gradient descent relative to a negative log likelihood (NLL) loss and Adam optimizer [20]. An initial learning rate of 10^{-2} was used, which was decreased by a factor of 10 each time the validation loss failed to improve for 5 consecutive epochs. In this manner, piecewise constant learning rates of 10^{-2}, 10^{-3}, and 10^{-4} were used, with the next drop in learning rate signalling the end of training.

The u-MPS was trained identically for all experiments, with the unidirectional LSTM trained in the same way. The bidirectional LSTM was trained specifically for the string completion task, with the loss taken as a sum of the NLL of the correct character at each site of the training strings, given knowledge of all characters on the other sites. For each pair of sampling and completion tasks in Table 3, the same trained u-MPS model was used to produce both statistics.