Structure-Aware Path Inference for Neural Finite State Transducers

Weiting Tan **Chu-Cheng Lin**[*] **Jason Eisner**
Department of Computer Science
Johns Hopkins University
`wtan12@jhu.edu {kitsing, jason}@cs.jhu.edu`

Abstract

Neural finite-state transducers (NFSTs) form an expressive family of neurosymbolic sequence transduction models. An NFST models each string pair as having been generated by a latent path in a finite-state transducer. As they are deep generative models, both training and inference of NFSTs require inference networks that approximate posterior distributions over such latent variables. In this paper, we focus on the resulting challenge of imputing the latent alignment path that explains a given pair of input and output strings (e.g., during training). We train three autoregressive approximate models for amortized inference of the path, which can then be used as proposal distributions for importance sampling. All three models perform lookahead. Our most sophisticated (and novel) model leverages the FST structure to consider the graph of future paths; unfortunately, we find that it loses out to the simpler approaches—except on an *artificial* task that we concocted to confuse the simpler approaches.

1 Introduction

Recent advances in applied deep learning have typically applied end-to-end training to homogeneous architectures such as recurrent neural nets (RNNs) [23], convolutional neural networks [11], or Transformers [27]. For small-data settings, however, end-to-end training can benefit from inductive bias through domain-specific constraints and featurization [20]. In the case of sequence-to-sequence problems—e.g., grapheme-to-phoneme [9] or speech-to-text [17]—one technique is to use finite-state transducers (FSTs) [10], whose topology can be manually designed based on the task of interest. In this paper, we design and compare inference networks for use with "neuralized" FSTs.

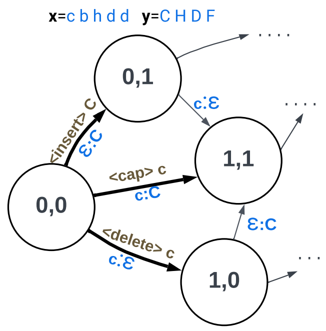

Figure 1: Marked finite-state transducer, all of whose paths generate input $\boldsymbol{x}$ and output $\boldsymbol{y}$. We show the transitions from the initial state, with their input:output symbols and marks. The same example appears in Fig. 3 without the marks shown.

Neuralized FSTs (NFSTs) [14] abandon the Markov property to become more expressive than standard arc-weighted FSTs [6]. A path's weight is computed by some arbitrary neural model from the ordered string of marks encountered along the path. The marks on each arc provide features of the transduction operation carried out by that arc (as illustrated in Fig. 1).

However, the expressiveness of NFSTs comes at the cost of training efficiency. Modeling joint or conditional probabilities on observed string pairs $(\boldsymbol{x}, \boldsymbol{y})$ requires imputing the latent NFST path $\boldsymbol{z}$ that aligns each observed input string $\boldsymbol{x}$ with its output string $\boldsymbol{y}$, as we review in §2. Summing over these

[*]Now at Google.

Proceedings of the workshop "I Can't Believe It's Not Better: Failure Modes in the Age of Foundation Models" at 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

paths is expensive because NFSTs give up the Markov property that enables dynamic programming on traditional weighted FSTs [16]. We must fall back on importance sampling. To this end, Lin et al. [14] built an autoregressive proposal distribution for these latent paths.[1]

Their proposal distribution—basically the SWS method described in §3.2 below—sampled a path through the NSFT from left to right. At each step, the choice of the next arc was influenced by the prefix path sampled so far and by the suffixes of $\boldsymbol{x}$ and $\boldsymbol{y}$ that have yet to be aligned. **In this paper, we extend this idea (§3.3) to consider the graph of possible alignments of those suffixes (Fig. 3), as determined by the NFST topology.** We evaluated the quality of the proposal distributions on three tasks:

- (`tr`) reverse transliteration of Urdu words from the Roman alphabet to the Urdu alphabet
- (`scan`) compositional navigation commands paired with the corresponding action sequences
- (`cipher`) synthetic dataset created by enciphering the input text with certain patterns

Task examples are shown in Table 1, and more descriptions are available in Appendix A. We compared our novel proposal distribution (§3.3) to the approach of [14] (§3.2) and to an even simpler baseline (§3.1). Overall, it was difficult to get our novel method to work. In the `tr` and `scan` tasks, it is apparently possible to choose the next arc well enough by the existing method of looking ahead to the *unaligned* suffixes. Perhaps the existing method learns to compare their lengths or their unordered bags of symbols. We designed the `cipher` task to frustrate such heuristics, and there our novel method really was necessary, benefiting from its domain knowledge of possible alignments (the given FST). But for the `tr` and `scan` tasks, our novel proposal distribution did considerably worse—perhaps our architecture was unnecessarily complicated and harder to train. This raises questions about the necessity and wisdom of explicitly considering the graph of possible alignments for real-world tasks.

2 Preliminaries: Neuralized Finite-State Transducers

2.1 Marked FSTs

A marked FST, or MSFT, is a directed graph in which some states are designated as initial and/or final, and each arc is labeled with an input substring, an output substring, and a mark substring. An *generating path* in the MFST is any path $\boldsymbol{z}$ from an initial state to a final state. It is said to generate the pair $(\boldsymbol{x}, \boldsymbol{y})$ with mark string $\boldsymbol{\omega}$ if $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega}$ respectively are the concatenations of the input, output, and mark substrings of $\boldsymbol{z}$'s arcs.[2]

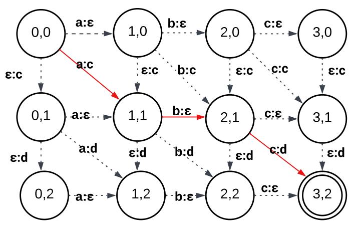

Figure 2: Directed graph constructed by composing an edit-distance MFST $\mathcal{T}$ with input $\boldsymbol{x} = abc$ and output $\boldsymbol{y} = cd$. The marks are suppressed here, but see Fig. 1.

The mark string on a generating path of $\mathcal{T}$ provides domain-specific information about the path. It may record information about the states along the path, the symbols being generated (for example, their phonetic or orthographic properties), how the path aligns input and output symbols (that is, which symbols or properties are being edited), and the contexts of these aligned symbols (for example, whether they fall in the onset, nucleus, or coda of a linguistic syllable). We may compose the MFST $\mathcal{T}$ with strings $\boldsymbol{x}, \boldsymbol{y}$ to obtain a restricted FST $\boldsymbol{x} \circ \mathcal{T} \circ \boldsymbol{y}$ whose generating paths correspond exactly to the paths in $\mathcal{T}$ that generate $(\boldsymbol{x}, \boldsymbol{y})$, with the same marks. A standard simple example is shown in Fig. 2: input "abc" and output "cd" have been composed with a 1-state MFST whose arcs (which are self-loops) allow symbol insertions, deletions, and substitutions. The generating paths are the paths from $(0, 0)$ to $(3, 2)$. The red path in Fig. 2 transforms "abc" to "cd" by deleting the second symbol of "abc" and substituting for the others.

2.2 Neuralized FSTs

A neuralized finite-state transducer (NFST) is an MFST $\mathcal{T}$ paired with some parametric scoring function $\tilde{p}_\theta$ that maps any generating path's mark string to a non-negative weight. Given an appropriate $\mathcal{T}$, this defines an unnormalized probability distribution $\tilde{p}_\theta$ over the paths. See Appendix B for a formal definition.

[1]Better ensembles of weighted proposals can be jointly generated by using multinomial resampling (in particle filtering or particle smoothing), as Lin and Eisner [13] did in a simpler setting. We do not pursue this extension here.

[2]An ordinary FST omits the mark string, and the familiar FSA also omits the output string.

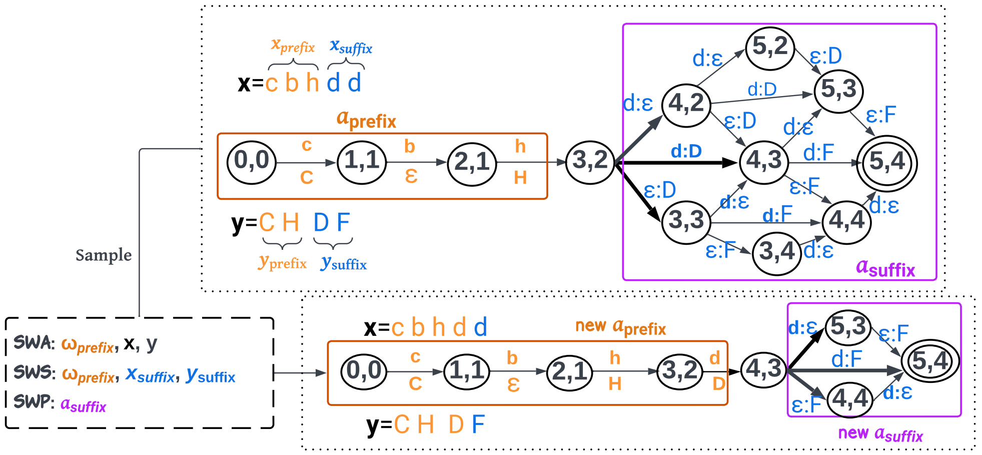

Figure 3: Choosing a generating path $\boldsymbol{a}$ in $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ with mark string $\boldsymbol{\omega}$. (Note that $\boldsymbol{a}$ ranges over paths in $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$, whereas $\boldsymbol{z}$ ranges over paths in $\mathcal{T}$. Marks are not shown.) The top graph shows that after choosing the first 3 arcs $\boldsymbol{a}_{\text{prefix}} = a_1a_2a_3$, reaching state $(3,2)$, the sampler must choose a suffix path through the subgraph $\boldsymbol{a}_{\text{suffix}}$ of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ that is reachable from $(3,2)$. Its choices for a_4 are the three thick out-arcs from $(3,2)$; choosing the arc to $(4,3)$ yields the reduced graph of possible paths $\boldsymbol{a}$ at the bottom. To make its stochastic choice, the sampler (SWA, SWS, or SWP) conditions on certain properties of the top graph, shown in the dashed box at the left. Here $\boldsymbol{x}_{\text{prefix}}, \boldsymbol{y}_{\text{prefix}}, \boldsymbol{\omega}_{\text{prefix}}$ refer to the labels on $\boldsymbol{a}_{\text{prefix}}$, while $\boldsymbol{x}_{\text{suffix}}, \boldsymbol{y}_{\text{suffix}}$ are the remaining portions of $\boldsymbol{x}, \boldsymbol{y}$. SWP's choice is determined by a global backward pass on $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ in which the probabilities of the out-arcs depend only on $\boldsymbol{a}_{\text{suffix}}$.

In practice, we will estimate $\tilde{p}_\theta$ by estimating its parameters θ. The unnormalized probability of a string pair is obtained by summing over the paths $\boldsymbol{z}$ that might have generated that string:

$$\tilde{p}_\theta(\boldsymbol{x},\boldsymbol{y}) \overset{\text{def}}{=} \textstyle\sum_{\boldsymbol{z}\in\boldsymbol{x}\circ\mathcal{T}\circ\boldsymbol{y}} \tilde{p}_\theta(\boldsymbol{z}) \tag{1}$$

Given the pair $(\boldsymbol{x},\boldsymbol{y})$, the posterior distribution over the latent generating path $\boldsymbol{z} \in \boldsymbol{x}\circ\mathcal{T}\circ\boldsymbol{y}$ is $p_\theta(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{y}) \overset{\text{def}}{=} \tilde{p}_\theta(\boldsymbol{z})/\tilde{p}_\theta(\boldsymbol{x},\boldsymbol{y})$. We emphasize that $\tilde{p}_\theta(\boldsymbol{z})$ depends on $\boldsymbol{z}$ only through its mark string.

Computing the quantity $\log\tilde{p}_\theta(\boldsymbol{x},\boldsymbol{y})$ is crucial for training θ. However, equation (1)'s marginalization over mark strings $\boldsymbol{z}$ is in general intractable. We resort to using a Monte Carlo variational lower bound, which imputes $\boldsymbol{z} \sim p_\theta(\cdot \mid \boldsymbol{x},\boldsymbol{y})$ by importance sampling using a neural proposal distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{y}) \approx p_\theta(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{y})$. In Appendix C we describe a procedure for jointly training p_θ and q_ϕ, making use of the importance weighting estimator [3] and making certain assumptions about $\mathcal{T}$ and p_θ.

Our focus in this paper is to consider different parametric forms for the distribution q_ϕ over paths that generate $(\boldsymbol{x},\boldsymbol{y})$. To simplify our study, we assume that θ is given, and only train ϕ to minimize the divergence $\mathrm{KL}(p_\theta \parallel q_\phi) = \underset{\boldsymbol{z}\sim p(\cdot\mid\boldsymbol{x},\boldsymbol{y})}{\mathbb{E}}[-\log q_\phi(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{y})]$ by following its gradient (or rather, the biased estimate of its gradient that we obtain by normalized importance sampling, sample size 16).

3 Three Proposal Distributions

We explore three distribution families $q_\phi(\boldsymbol{z} \mid \boldsymbol{x},\boldsymbol{y})$, sketched in Fig. 3. Like p_θ itself, each q_ϕ family is insensitive to the specific topology and labeling of $\mathcal{T}$. Any MFST that was *equivalent* to $\mathcal{T}$ in the sense of generating the same set of $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\omega})$ triples—that is, the same regular 3-way relation—would give the same parametric proposal distribution q_ϕ. Sampling from q_ϕ in each case is done by sampling a mark string $\boldsymbol{\omega}$ and using it to identify a path $\boldsymbol{z}$ in $\mathcal{T}$. To make this identification possible, we henceforth assume that distinct paths in $\mathcal{T}$ with the same $\boldsymbol{x},\boldsymbol{y}$ always have distinct mark strings. (A stronger assumption is already needed for the particular model $\tilde{p}_\theta$ that we spell out in Appendix C.)

Let $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ be a version of $\boldsymbol{x}\circ\mathcal{T}\circ\boldsymbol{y}$ that has been determinized with respect to the mark tape and then minimized.[3] Then $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is a canonical MFST that expresses the same relation as $\boldsymbol{x}\circ\mathcal{T}\circ\boldsymbol{y}$, while

[3]Brief technical details [see e.g. 21, 15, 1]: Treat $\mathcal{T}$ as a finite-state automaton over the mark alphabet, weighted by (input, output) string pairs. $\boldsymbol{x}\circ\mathcal{T}\circ\boldsymbol{y}$ can be determinized (made subsequential) because it is

guaranteeing that (a) each arc's mark substring has length 1, (b) the out-arcs from a state are labeled with different marks, and (c) every choice of out-arc can lead to a final state.

It follows that we can sample a mark string $\boldsymbol{\omega}$ by autoregressively sampling a generating path $\boldsymbol{a}$ in $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$, at each step requiring only a distribution over the out-arcs from the current state, or equivalently, over their distinct marks. We abuse notation and write q_ϕ for $q_\phi(\boldsymbol{a} \mid \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}})$, $q_\phi(\boldsymbol{\omega} \mid \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}})$, and $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$.

Since we use the standard minimization construction, $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is *reduced*, meaning that the input and output symbols on a path appear "as soon as possible" [21]. This standardizes how the symbols are distributed along the path, ensuring that $\boldsymbol{x}_{\text{suffix}}, \boldsymbol{y}_{\text{suffix}}$ are well-defined in the SWS sampler (§3.2).

3.1 Sampler with Attention (SWA)

The SWA sampler proposes the next mark ω_t in $\boldsymbol{\omega}$ by considering (1) the marks $\boldsymbol{\omega}_{<t} = \omega_1 \cdots \omega_{t-1}$ already chosen and (2) an assessment of which marks might be chosen in future. For (1), we use a recurrent neural network to encode $\boldsymbol{\omega}_{<t}$ into a vector $\mathbf{h}_{t-1}$. For (2), an attention mechanism is employed to mine information from the raw strings $(\boldsymbol{x}, \boldsymbol{y})$. More formally, SWA samples from

$$q_\phi(\omega_t \mid \boldsymbol{\omega}_{<t}, \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}) \propto \exp(W\,[1; \mathbf{h}_{t-1}; \text{Att}(\mathbf{h}_{t-1}, \text{enc}(\boldsymbol{x}, \boldsymbol{y}))]) \tag{2}$$

meaning a softmax distribution over just the marks ω_t that $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ allows to follow $\boldsymbol{\omega}_{<t}$. Equation (2) applies the learned matrix W to the concatenation of the state $\mathbf{h}_{t-1}$, which encodes $\boldsymbol{\omega}_{\text{prefix}}$, and $\text{Att}(\cdots)$ which uses $\mathbf{h}_{t-1}$ to attend to $(\boldsymbol{x}, \boldsymbol{y})$ (not necessarily just $\boldsymbol{x}_{\text{suffix}}, \boldsymbol{y}_{\text{suffix}}$ in Fig. 3 We compute $\mathbf{h}_{t-1} \in \mathbb{R}^d$ with a left-to-right GRU [4], as in Lin and Eisner [13]. We compute $\text{enc}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{n\times d}$ by applying a separate bidirectional GRU[4] to the concatenation $\boldsymbol{x}\#\boldsymbol{y}^R$, whose length we denote by n.[5] We then use $\mathbf{h}_{t-1}$ as a softmax-attention query over the n encoded tokens of $\boldsymbol{x}\#\boldsymbol{y}^R$:

$$\text{Att}(\mathbf{h}_{t-1}, \text{enc}(\boldsymbol{x}, \boldsymbol{y})) = \sum_{i=1}^{n} a_i\, \text{enc}_i(\boldsymbol{x}, \boldsymbol{y}), \quad \text{where } a_i = \frac{\exp(\mathbf{h}_{t-1} \cdot \text{enc}_i(\boldsymbol{x}, \boldsymbol{y}))}{\sum_{j=1}^{n} \exp(\mathbf{h}_{t-1} \cdot \text{enc}_j(\boldsymbol{x}, \boldsymbol{y}))} \tag{3}$$

3.2 Sampler with State Tracking (SWS)

The SWS sampler is simpler than SWA (no attention), but it takes care to consider only the suffixes of $\boldsymbol{x}$ and $\boldsymbol{y}$ that remain to be aligned. The prefix path of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ marked with $\boldsymbol{\omega}_{<t}$ has generated aligned input and output strings $\boldsymbol{x}_{<t}, \boldsymbol{y}_{<t}$ (which may not have length t). Let $\boldsymbol{x}_{\geq t}$ and $\boldsymbol{y}_{\geq t}$ denote the unaligned suffixes of $\boldsymbol{x}, \boldsymbol{y}$, and encode them into vectors by enc_x and enc_y, which are separate right-to-left GRUs. SWS samples from

$$q_\phi(\omega_t \mid \boldsymbol{\omega}_{<t}, \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}) \propto \exp(W\,[1; \mathbf{h}_{t-1}; \text{enc}_\text{x}(\boldsymbol{x}_{\geq t}); \text{enc}_\text{y}(\boldsymbol{y}_{\geq t})]) \tag{4}$$

As before, this is a softmax over choices of ω_t that are legal under $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$; other marks have probability 0.

The SWS method is directly inspired by [14]. It is a relatively weak model, as the three information sources $\boldsymbol{\omega}_{<t}$, $\boldsymbol{x}_{\geq t}$, and $\boldsymbol{y}_{\geq t}$ contribute *independent* summands to the logits of the possible marks. There is no additional feed-forward layer that allows these sources to interact. In contrast, SWA mixed all three sources by having an encoding of $\boldsymbol{\omega}_{<t}$ attend to a joint encoding of $\boldsymbol{x}\#\boldsymbol{y}^R$.

3.3 Sampler with Path Structures (SWP)

The main contribution of this work is the SWP sampler. It assigns embeddings and weights to the arcs of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ and uses this to define a distribution over its paths $\boldsymbol{a}$, yielding a distribution over mark strings $\boldsymbol{\omega}$. So it samples $\boldsymbol{a} \in \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ and then returns the $(\boldsymbol{x}, \boldsymbol{y})$-generating path $\boldsymbol{z} \in \mathcal{T}$ with the same marks.

SWP's proposal distribution assigns weights to the arcs of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ and **treats it as a weighted FST (WFST)**. Recall that in the NFST $\mathcal{T}$, paths are scored globally using $\tilde{p}_\theta(\boldsymbol{z})$. A WFST is simpler: each

unambiguous (due to our assumption above) and has bounded variation (since all mark strings map to the fixed strings $\boldsymbol{x}$ and $\boldsymbol{y}$) or equivalently has the twins property (since for the same reason, all cycles must produce empty input and output). We remark that only the SWS sampler really requires the (input, output) weights, as they guide its sampling of a mark string. The SWA and SWP samplers could simply drop them from $\boldsymbol{x} \circ \mathcal{T} \circ \boldsymbol{y}$ and apply ordinary unweighted determinization and minimization [see 8].

[4]Other representation methods like Transformer encoder could also be used.

[5]Inspired by [26], we use the reversed output string $\boldsymbol{y}^R$ so that the end of $\boldsymbol{x}$ and the end of $\boldsymbol{y}$ are adjacent. This setup makes it potentially easier for $\text{enc}(\boldsymbol{x}, \boldsymbol{y})$ to consider alignments between $\boldsymbol{x}_{\text{suffix}}$ and $\boldsymbol{y}_{\text{suffix}}$.

arc $s \to s'$ has a fixed weight $w_{s \to s'}$, and the weight of a path $\boldsymbol{a}$ is the product of its arc weights.[6] This makes exact path sampling tractable: if the path $\boldsymbol{a}_{<t} = a_1 \cdots a_{t-1} =$ ends at state s, then the autoregressive probability of choosing a_t to be the arc $s \to s'$ depends only on s (a Markov property):

$$q_\phi(s \to s' \mid \boldsymbol{a}_{<t}, \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}) = q_\phi(s \to s' \mid s) = w_{s \to s'} \beta(s') \,/\, \beta(s) \tag{5}$$

where $\beta(s)$ denotes the "backward weight"—the total weight of all paths from s to a final state.

The $\beta(s)$ values are the solution to the system of linear equations $\beta(s) = (\sum_{s'} w_{s \to s'} \beta(s')) + \mathbb{I}(s \text{ is final})$. If $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is acyclic, this is an acyclic recurrence that can be solved in linear time by the backward algorithm [2, 19].

But where do the WFST arc weights $w_{s \to s'}$ come from? We will assume $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is acyclic and use a similar recurrence to define arc embeddings $\mathbf{e}_{s \to s'}$ and state embeddings $\mathbf{e}_s$ that yield the arc weights $w_{s \to s'}$. Algorithm 1 computes these along with $\beta(s)$ and $q_\phi(s \to s' \mid s, \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}})$, all in a single pass.

The state embedding $\mathbf{e}_s$ is a LatticeRNN-like embedding [25] that attempts to summarize the mark strings of all suffix paths from s. A graph of such paths is shown as $\boldsymbol{a}_{\text{suffix}}$ in Fig. 3. Suffix paths with higher WFST weight will have more influence on the summary, thanks to the weighted average at line 8. Similarly, the arc embedding $\mathbf{e}_{s \to s'}$ attempts to summarize all suffix paths of the form $s \to s' \to \cdots$. Put another way, it encodes the mark on $s \to s'$ in a way that considers its possible right contexts.

Algorithm 1 Constructing WFST arc weights w, backward weights β, and transition probabilities q

```
   Input: An acyclic MFST $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$,
1: for $s \leftarrow$ states($\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$) in reverse topological order :
2:     $\beta(s) = \mathbb{I}(s \text{ is final})$
3:     for each out-arc $s \to s'$ (with mark $\omega$) :              ▷ note that $s'$ has already been visited
4:         $\mathbf{e}_{s \to s'} = \sigma(U\,[1; \mathbf{e}_\omega; \mathbf{e}_{s'}])$      ▷ $U$ and mark embeddings $\mathbf{e}_\omega$ are learned (part of $\phi$)
5:         $w_{s \to s'} = \exp(\mathbf{w} \cdot \mathbf{e}_{s \to s'})$                        ▷ arc weight; $\mathbf{w}$ is learned
6:         $\beta(s) \mathrel{+}= w_{s \to s'} \beta(s')$                                  ▷ backward algorithm
7:     $q_\phi(s \to s' \mid s) = w_{s \to s'} \beta(s') / \beta(s)$                ▷ transition probability (equation (5))
8:     $\mathbf{e}_s = \sum_{s'} q_\phi(s \to s' \mid s)\, \mathbf{e}_{s \to s'}$   ▷ state embedding is weighted average of arc embeddings
9: return all transition probabilities $q_\phi(s \to s' \mid s)$
```

After running Algorithm 1, we can use q_ϕ to sample a path $\boldsymbol{a}$ of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ from left to right, yielding its mark string $\boldsymbol{\omega}$. Choosing the next arc a_t is tantamount to choosing its mark as the next mark ω_t.

This formulation of SWP does have some limitations relative to SWA and SWS. The right-to-left embedding update at line 4 is RNN-style; one would have to add a GRU-style forget gate to make it comparable. Another limitation of SWP is that its choice of ω_t is *not* influenced by the embedding $\mathbf{h}_{t-1}$ of $\boldsymbol{\omega}_{t-1}$. An obvious fix would be to replace $\mathbf{w}$ with $\mathbf{h}_{t-1}$ at line 5. However, then sampling at each step t would require re-running Algorithm 1 with the new $\mathbf{h}_{t-1}$ to re-embed and re-weight the sub-MFST that is reachable from the current state s (shown in Fig. 3). A cheaper variant would run Algorithm 1 only at the start and never change the embeddings or weights, but have step t choose arc $s \to s'$ with probability proportional to $\exp(\mathbf{h}_{t-1} \cdot \mathbf{e}_{s \to s'})\beta(s')$ or perhaps just $\exp(\mathbf{h}_{t-1} \cdot \mathbf{e}_{s \to s'})$.

4 Training and Evaluation of Proposal Samplers

Recall that our q_ϕ chooses $\boldsymbol{z}$ in $\mathcal{T}$ (by sampling $\boldsymbol{\omega}$ from $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$). To evaluate whether $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) \approx p_\theta(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$ as desired, we follow Lin and Eisner [13] and consider the *exclusive* KL divergence

$$\mathrm{KL}(q_\phi \parallel \tilde{p}_\theta) = \mathbb{E}_{\boldsymbol{z} \sim q_\phi(\cdot \mid \boldsymbol{x}, \boldsymbol{y})}[\log q_\phi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) - \log p_\theta(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})] \tag{6}$$

However, since the normalization term (1) is hard to compute, we drop it and compute the resulting *Partial KL*, averaging it over a held-out test dataset $\mathcal{D}$ (and estimating the expectation by sampling):

$$\frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}} \mathbb{E}_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x}_i, \boldsymbol{y}_i)} \left[\log q_\phi(\boldsymbol{z} \mid \boldsymbol{x}_i, \boldsymbol{y}_i) - \log \tilde{p}_\theta(\boldsymbol{z})\right] \tag{7}$$

To make the comparison fair, we need to enforce that the dropped normalization term (1) is the same for different samplers. Thus, we evaluate our different samplers with the same *frozen* model

[6]In general there could be multiple $s \to s'$ arcs, but for simplicity, we gloss over this in the notation.

$\tilde{p}_\theta$, only training the sampler parameters ϕ. We train ϕ to minimize the *inclusive* KL divergence, as mentioned at the end of §2.2, to avoid the instability of minimizing the exclusive divergence (7) by the high-variance REINFORCE estimator. The reason that we still evaluate with *exclusive* KL is that it serves our actual goals for q_ϕ: if we were training θ to minimize the variational log-likelihood (13) in Appendix C, then a q_ϕ with smaller exclusive KL would obtain a tighter variational bound.

We obtained our frozen $\tilde{p}_\theta$ by alternately optimizing θ (via (13)) and the ϕ of an SWA sampler q_ϕ (again trained via inclusive KL, for stability, even though exclusive KL would give a tighter bound (13)), simplifying the latter to replace GRUs with RNNs. We then discarded this sampler. (One might expect this choice of $\tilde{p}_\theta$ to benefit the SWA sampler, but as we will see below, it never performed best.)

5 Experiments

We conducted our experiments on three datasets (described in Appendix A): `tr`, `scan`, and `cipher`. Figure 4a has the main results. Figure 4b shows the importance sampling estimate of the expected mark string length $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathbb{E}_{\boldsymbol{z}\sim p_\theta(\cdot|\boldsymbol{x},\boldsymbol{y})}\left[|\boldsymbol{z}_\Omega|\right]\right]$ and Table 7 shows the (de-duplicated) effective sample size. For `tr` and `scan` tasks, the mark string length is fixed regardless of alignment, as the FST design only allows deletion/insertion. For `cipher`, however, SWP is able to discover higher-probability alignments with more substitutions and shorter mark strings, both by identifying the correct cipher and by aligning the strings more accurately under that cipher. This improves both *Partial KL* and estimated length. Qualitative examples of proposals from different samplers are presented in Appendix E.

Sampler	TR	Scan	Cipher	Avg
SWA	23.3	38.1	101.3	54.3
SWS	**21.8**	**25.4**	87.1	**44.7**
SWP	38.1	102.9	**73.9**	71.7

(a) Figure 4a. *Partial KL* divergence comparison (lower values are better) for different proposal samplers.

Sampler	TR	Scan	Cipher
SWA	24.3	65.6	68.4
SWS	24.3	65.6	67.6
SWP	24.3	65.6	59.9

(b) Figure 4b. Expected mark string length under p_θ, estimated by importance sampling with proposals $\sim q_\phi$.

For `tr`, the true alignment between Urdu and English scripts is monotonic, and the training/test sets exhibit similar length distributions. Here the SWA sampler performed comparably to the SWS sampler, suggesting that a basic attention mechanism suffices on this task. SWP showed no advantage in this case (indeed, it did worse), probably because simple symbol-counting and local-lookahead heuristics are effective at guessing the next step, and these can be captured by both SWA and SWS architectures.

In the `scan` dataset, where the true alignment is non-monotonic and test examples are longer than training examples, the SWS sampler outperforms the SWA sampler. Presumably SWS is able to find better alignments—that is, generating paths with higher scores $\tilde{p}(\boldsymbol{z})$—by looking ahead into the near future of the unaligned suffixes $\boldsymbol{x}_{\geq t}, \boldsymbol{y}_{\geq t}$ to see how to complete the partial alignment summarized by $\mathbf{h}_{t-1}$. We had expected that SWP might do even better by globally aligning $\boldsymbol{x}, \boldsymbol{y}$ using a WFST, but evidently it completely failed to learn the alignment (see Table 5), perhaps due to uninformed initialization[7] or the WFST's Markov property that ignores $\boldsymbol{z}_{<t}$. Indeed, SWP did even worse than a baseline "no-lookahead" model (see Fig. 7) we trained to predict the next mark ω_t from only $\mathbf{h}_{t-1}$.

Moving to the `cipher` task, however, the SWP sampler excelled. The FST's topology is more informed in this case, and plays a crucial role in considering possible alignments of $\boldsymbol{x}_{\geq t}, \boldsymbol{y}_{\geq t}$ (especially at $t = 1$, to identify which cipher should be used). The other samplers struggle here without this, as simple heuristics no longer work. A hyperparameter sweep for this dataset (see Fig. 6) reinforces the importance of alignment information: even the smallest model for the SWP sampler (hidden dimension=64) outperforms SWS and SWA samplers based on much larger models.

6 Conclusion

In this paper, we explored proposal distributions over paths in a graph. We found that while the proposal distribution could sometimes benefit from examining the structure of the graph, our proposal distribution that did so proved hard to train. As a result, except on a particularly challenging

[7]It might have helped to do *supervised* pre-training of q_ϕ on $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega})$ triples drawn unconstrained from $\tilde{p}_\theta$.

artificial task, our structure-aware sampler performed worse than its less complex, non-structure-aware alternatives. One might try improved training methods to minimize either inclusive KL, such as DPG_{off} [18], or exclusive KL, such as PPO with entropy bonus [24]. It is also possible that variant architectures would succeed better, as discussed in §3.3.

References

[1] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFST: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer.

[2] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41(1):164–171.

[3] Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders.

[4] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

[5] Du, L., Hennigen, L. T., Pimentel, T., Meister, C., Eisner, J., and Cotterell, R. (2023). A measure-theoretic characterization of tight language models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 9744–9770.

[6] Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8.

[7] Fisher, R. A. and Yates, F. (1938). Statistical tables for biological, agricultural and medical research.

[8] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA.

[9] Knight, K. and Graehl, J. (1998). Machine transliteration. *Computational Linguistics*, 24(4):599–612.

[10] Kornai, A., editor (1999). *Extended Finite State Models of Language*. Cambridge University Press.

[11] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[12] Leviathan, Y., Kalman, M., and Matias, Y. (2022). Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

[13] Lin, C.-C. and Eisner, J. (2018). Neural particle smoothing for sampling from conditional sequence models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 929–941, New Orleans, Louisiana. Association for Computational Linguistics.

[14] Lin, C.-C., Zhu, H., Gormley, M. R., and Eisner, J. (2019). Neural finite-state transducers: Beyond rational relations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 272–283, Minneapolis, Minnesota. Association for Computational Linguistics.

[15] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2).

[16] Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350.

[17] Mohri, M., Pereira, F., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer.

[18] Parshakova, T., Andreoli, J.-M., and Dymetman, M. (2019). Distributional reinforcement learning for energy-based sequential models. In *Proceedings of the NeurIPS Workshop on Optimization Foundations for Reinforcement Learning (OptRL*.

[19] Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[20] Rastogi, P., Cotterell, R., and Eisner, J. (2016). Weighting finite-state transductions with neural context. In *NAACL*.

[21] Reutenauer, C. (1990). Subsequential functions: Characterizations, minimization, examples. In Dassow, J. and Kelemen, J., editors, *Aspects and Prospects of Theoretical Computer Science*, pages 62–79, Berlin, Heidelberg. Springer.

[22] Roark, B., Wolf-Sonkin, L., Kirov, C., Mielke, S. J., Johny, C., Demirşahin, I., and Hall, K. (2020). Processing South Asian languages written in the Latin script: the Dakshina dataset. In *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*, pages 2413–2423.

[23] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation.

[24] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[25] Sperber, M., Neubig, G., Niehues, J., and Waibel, A. (2017). Neural lattice-to-sequence models for uncertain inputs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1380–1389.

[26] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.

[27] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Supplementary Material

Appendix Sections	Contents
Appendix A	Description of datasets
Appendix B	Definition of Neuralized Finite State Transducers (NFSTs)
Appendix C	Parametrizing and training NFST models
Appendix D	Hyperparameter settings
Appendix E	Qualitative comparison of samplers
Appendix F	Comparison with no-lookahead baseline sampler
Appendix G	Discussion of limitations

A Datasets

In this section, we describe the datasets in more detail. Examples of the 3 tasks are shown in Table 1 and statistics in Table 2.

Dataset	Input	Output
`tr`	a s a m a r	(some urdu text)
`scan`	look right twice after jump left	I_TURN_LEFT I_JUMP I_TURN_RIGHT I_LOOK I_TURN_RIGHT I_LOOK
`cipher`	g c c i n f j n c g	i i r d i t y s

Table 1: Examples of datasets used.

Task	Split	Input Length	Output Length	Avg #States	Avg #Arcs
tr, vocab size=177	Train	7.2	6.9	232	303
	Valid	7.3	7.0	237	310
	Test	3.0	6.8	220	287
scan, vocab size=35	Train	8.1	12.0	285	366
	Valid	8.0	11.5	271	349
	Test	9.0	25.3	658	854
cipher, vocab size=67	Train	6.2	6.4	1028	1191
	Valid	6.1	6.3	992	1149
	Test	12.0	12.2	3989	4645

Table 2: Data statistics for different tasks and splits. The vocab size refers to the size of the mark alphabet. The state and arc counts refer to $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ before minimization, averaged over $(\boldsymbol{x},\boldsymbol{y})$ pairs in the dataset. Note that a single edit operation such as "delete a" may be carried out in $\mathcal{T}$ by a single arc with input a, output ε, and mark string "<delete><a>", but each instance of this operation in $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ will require two arcs with an intermediate state, since each arc of $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ has exactly one mark. Before minimization, such intermediate states have only one out-arc, which reduces the arc-to-state ratio.

Transliteration (`tr`) For the reverse transliteration task, we use the English-Urdu section from the Dakshina [22] dataset. The topology is shown in Fig. 5a, which is a deletion/insertion MFST. Such an MFST allows for any arbitrary transliteration from the Roman alphabet Σ to the Urdu alphabet Δ. Examples are shown in Tables 1 and 4.

We hope that after training, the model prefers—and the sampler proposes—paths where corresponding English and Urdu tokens are deleted and inserted close to each other. Note that our MFST $\mathcal{T}$ does not include substitution arcs like $a : bc$. It would accomplish this substitution by a sequence such as $a : \varepsilon, \varepsilon : b, \varepsilon : c$, and $\tilde{p}_\theta$ can be trained to favor such sequences. This saves us from the need to enumerate all reasonable substitutions within $\mathcal{T}$. We do not even include copy arcs like $a : a$, since the Urdu and English alphabets are different.

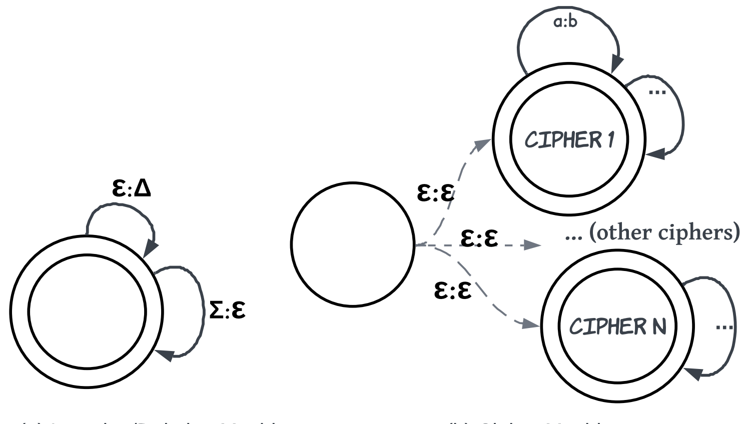

Figure 5: MFST topologies $\mathcal{T}$ used for different tasks. Here Σ and Δ are the alphabets of possible input and output symbols. An arc with input, output, and mark substrings $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega}$ would ideally be displayed with the label $\boldsymbol{x} : \boldsymbol{y}/\boldsymbol{\omega}$, but in this figure we suppress the marks $\boldsymbol{\omega}$.
(a) `tr` and `scan` only use this deletion/insertion topology, which freely generates input symbols $a \in \Sigma$ (via deletion arcs $a : \varepsilon$) interleaved arbitrarily with output symbols $b \in \Delta$ (via insertion arcs $\varepsilon : b$). Thus, it generates all $(\boldsymbol{x}, \boldsymbol{y})$ pairs with all possible alignments.
(b) `cipher` begins with the random-cipher MFST shown in the drawing. That MFST deterministically replaces each input symbol $a \in \Sigma$ with a specific other symbol $b \in \Delta$ (via a substitution arc $a : b$), according to a cipher selected nondeterministically from $N = 5$ possible ciphers. The `cipher` task composes this with a deletion/insertion/copy MFST, which has a single state (just as in the `tr` task) and self-loop arcs of the form $b : \varepsilon$ (deletion), $\varepsilon : b$ (insertion), and $b : b$ (copy) for all $b \in \Delta$. Thus, some of the enciphered input symbols are deleted from the output, while some are copied and some other output symbols are inserted. We implement the composition of two MFSTs so that when $a : b$ with marks $\boldsymbol{\omega}_1$ from the cipher MFST composes with $b : \varepsilon$ or $b : b$ with marks $\boldsymbol{\omega}_2$ from the deletion/insertion/copy MFST, the resulting arc (whose input:output is $a : \varepsilon$ or $a : b$ respectively) will have marks $\boldsymbol{\omega}_1\boldsymbol{\omega}_2$, in that order.

Navigation Commands (`scan`) We apply the same deletion/insertion topology to the `scan` dataset. The dataset contains command instructions in an English-like artificial language as input and action sequences as output. Alignments are nonmonotonic. Examples are shown in Tables 1 and 5.

In this dataset, test samples are longer on average than training samples, requiring the model $\tilde{p}_\theta$ to actually learn the correspondence between commands and actions to generalize well to test data. Whether or not $\tilde{p}_\theta$ succeeds in generalization, our goal in this paper is to make q_ϕ approximate the conditionalization of $\tilde{p}_\theta$ on test data.

Cipher (`cipher`) We synthetically create a cipher dataset by composing a cipher MFST with an deletion/insertion/copy MFST (see Fig. 5). Each of the 5 ciphers is chosen randomly (through random permutation using the Fisher-Yates shuffle algorithm [7]). As shown in Fig. 5b, the initial state has ε arcs to 5 cipher states; these arcs are marked with distinct marks. The i^{th} cipher state has 26 arcs corresponding to the enciphering process $x \to \sigma_i(x)$ for an input alphabet of size 26, and the i^{th} cipher function $\sigma_i(\cdot)$ is chosen randomly. As explained in the caption to Fig. 5, we then compose the cipher MFST with a deletion/insertion/copy MSFT that introduces noise into the ciphertext. The point of this construction is that a cipher-dependent alignment is now required. Similar to `scan`, we intentionally make the training samples shorter on average compared to the test samples, so that $\tilde{p}_\theta$ must learn to generalize.

Example mark strings are shown in Table 6. Note that the sampler q_ϕ must begin by predicting a mark that indicates which of the 5 ciphers will be used. All of our sampler designs use lookahead to make this prediction, but SWP has an advantage, because it can examine the entire graph $\boldsymbol{a}_{\text{suffix}}$ of possible $\boldsymbol{x}$-to-$\boldsymbol{y}$ alignments under all 5 ciphers. This graph is constructed with explicit knowledge of the FST topology, which the other samplers lack.

B Neuralized Finite-State Transducers

A neuralized finite state transducer (NFST) [14] is defined as a pair $(\mathcal{T}, \tilde{p}_\theta)$ where $\mathcal{T}$ is a marked finite-state transducer (MFST) and $\tilde{p}_\theta$ is a mark string scoring function parmeterized by θ.

An MFST can be regarded as having three left-to-right tapes, one for input symbols, one for output symbols, and one for the mark symbols. More formally,

$$\mathcal{T} = (\Sigma, \Delta, \Omega, Q, E, q_{\text{init}}, F)$$

where Σ, Δ, Ω are alphabets of input symbols, output symbols, and mark symbols. Q is a finite set of states and $E \subseteq Q \times (\Sigma^* \times \Delta^* \times \Omega^*) \times Q$ is a finite multiset of arcs. q_{init} is the initial state and F is a set of final states. Note that the marks on an arc may mention the input and output on that arc, but they can also mention other features that are consumed by $\tilde{p}_\theta$; for example, the symbol "<C>" can be used to mark every arc that outputs a consonant. A generating path $\boldsymbol{z}$ in $\mathcal{T}$ (also known as an accepting path) is a sequence of arcs from E that forms a path from q_{init} to any state in F. We write $\boldsymbol{z}_\Sigma$, $\boldsymbol{z}_\Delta$, or $\boldsymbol{z}_\Omega$ respectively for the concatenation of the input, output, or mark strings along the arcs of $\boldsymbol{z}$.

The mark scoring function $\tilde{p}_\theta : \Omega^* \to \mathbb{R}_{\geq 0}$ maps mark strings to non-negative scores. Define

$$Z_\theta \overset{\text{def}}{=} \sum_{\boldsymbol{z} \in \mathcal{T}} \tilde{p}_\theta(\boldsymbol{z}_\Omega) \tag{8}$$

$$\tilde{p}_\theta(\boldsymbol{x}, \boldsymbol{y}) \overset{\text{def}}{=} \sum_{\boldsymbol{z} \in \mathcal{T}:\ \boldsymbol{z}_\Sigma = \boldsymbol{x}, \boldsymbol{z}_\Delta = \boldsymbol{y}} \tilde{p}_\theta(\boldsymbol{z}_\Omega) \tag{9}$$

where $\boldsymbol{z} \in \mathcal{T}$ ranges over generating paths of $\mathcal{T}$. Provided that Z_θ is finite and positive, the NFST defines a parametric probability distribution over the generating paths of $\mathcal{T}$, namely $p_\theta(\boldsymbol{z}) \overset{\text{def}}{=} \tilde{p}_\theta(\boldsymbol{z})/Z_\theta$. It also defines a parametric probability distribution over $\Sigma^* \times \Delta^*$, namely $p_\theta(\boldsymbol{x}, \boldsymbol{y}) \overset{\text{def}}{=} \tilde{p}_\theta(\boldsymbol{x}, \boldsymbol{y})/Z_\theta$.

C Parametrizing and Training NFST

Given an NFST $(\mathcal{T}, \tilde{p}_\theta)$ and a dataset $\mathcal{D}$ of $(\boldsymbol{x}, \boldsymbol{y})$ pairs, a natural estimator to use for θ is the maximum likelihood estimator:

$$\hat{\theta}_\mathcal{D} = \arg\max_\theta \left\{ \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} [\log \tilde{p}_\theta(\boldsymbol{x}, \boldsymbol{y}) - \log Z_\theta] \right\} \tag{10}$$

The difficulty is in computing Z_θ and ensuring that it is finite. We can avoid these difficulties by dropping the $\log Z_\theta$ term. That would be justified if $Z_\theta = 1$; a first attempt to guarantee that is to adopt a locally normalized (autoregressive) model of mark strings:

$$\tilde{p}_\theta(\boldsymbol{\omega}) = \prod_{t=1}^{T+1} p_\theta(\omega_t \mid \boldsymbol{\omega}_{<t}) \tag{11}$$

where $T = |\boldsymbol{\omega}|$, ω_{T+1} by convention is a distinguished end-of-sequence symbol $\textsc{eos} \notin \Omega$, and any parametric conditional probability distribution over $\Omega \cup \{\textsc{eos}\}$ may be used for the factors. This often ensures that $\sum_{\boldsymbol{\omega} \in \Omega^*} \tilde{p}_\theta(\boldsymbol{\omega}) = 1$. However, it does not ensure $Z_\theta = 1$ as desired.

To avoid $Z_\theta > 1$, we will require that different paths of $\mathcal{T}$ have different mark strings, so that no string in Ω^* is double-counted. However, typically $Z_\theta < 1$ (the distribution is deficient), since some probability mass is allocated to "ungrammatical" mark strings that do not appear on any path of $\mathcal{T}$.[8]

By regarding the set of ungrammatical mark strings as a special event $\perp$, we may regard $\tilde{p}_\theta$ as a probability distribution over $\Omega^* \cup \{\perp\}$. As *that* distribution is already normalized ($Z_\theta + \tilde{p}_\theta(\perp) = 1$), its maximum likelihood estimator is

$$\hat{\theta}_\mathcal{D} = \arg\max_\theta \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \log \tilde{p}_\theta(\boldsymbol{x}, \boldsymbol{y}) \tag{12}$$

[8]For some families of conditional distributions in equation (11), it is even possible for some probability mass to be allocated to infinite mark sequences [5], which we also regard as ungrammatical. (Then even $\sum_{\boldsymbol{\omega} \in \Omega^*} \tilde{p}_\theta(\boldsymbol{\omega}) < 1$.)

In practice, $p_{\hat{\theta}_{\mathcal{D}}}$ will tend to place a positive but small probability on $\perp$, since $\perp$ is never observed in $\mathcal{D}$. The maximum likelihood objective would prefer to raise the probability of grammatical mark strings and particularly the ones that can explain $\mathcal{D}$. In other words, training θ in this way should approximately learn the grammaticality constraint $\tilde{p}_\theta(\perp) \approx 0$ rather than having it be a structural zero of the model. In effect, it forces θ to learn something about the structure of $\mathcal{T}$ (which could, in fact, serve as a useful form of multi-task regularization that improves generalization).

The remaining difficulty is that the log-likelihood (12) requires an intractable sum over all paths that generate $(\boldsymbol{x}, \boldsymbol{y})$. So instead of minimizing the negative log-likelihood, we minimize a variational upper bound (from Jensen's inequality) that can be estimated by sampling:

$$\begin{aligned}
\mathcal{L}(\theta) &\stackrel{\text{def}}{=} -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\log \tilde{p}_\theta(\boldsymbol{x},\boldsymbol{y})\right] \\
&= -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\log \sum_{\boldsymbol{z}\in\mathcal{T}:\ \boldsymbol{z}_\Sigma=\boldsymbol{x},\boldsymbol{z}_\Delta=\boldsymbol{y}} \tilde{p}_\theta(\boldsymbol{z})\right] \\
&= -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\log \mathbb{E}_{\boldsymbol{z}\sim q_\phi}\left[\frac{\tilde{p}_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x},\boldsymbol{y})}\right]\right] \\
&\le -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathbb{E}_{\boldsymbol{z}\sim q_\phi}\left[\log \frac{\tilde{p}_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x},\boldsymbol{y})}\right]\right] \\
&\stackrel{\text{def}}{=} \mathcal{L}'(\theta)
\end{aligned} \tag{13}$$

where $q_\phi(\cdot \mid \boldsymbol{x}, \boldsymbol{y})$ may be any conditional distribution over the generating paths of $(\boldsymbol{x}, \boldsymbol{y})$ in $\mathcal{T}$. The tightest bound for a given parametric family q_ϕ is obtained by minimizing $\mathcal{L}'(\theta)$ as a function of ϕ, which minimizes the exclusive KL divergence (6).

For any q_ϕ, we may use the importance weighting estimator IWAE [3] to obtain an even tighter upper bound based on averaging over $K > 1$ paths under the log:

$$\mathcal{L}'_K(\theta) \stackrel{\text{def}}{=} -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{\boldsymbol{z}^{(1)}\ldots\boldsymbol{z}^{(K)}\sim q_\phi}\left[\log \frac{1}{K}\sum_{k=1}^{K}\frac{\tilde{p}_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x},\boldsymbol{y})}\right]\right] \tag{14}$$

which can again be estimated by sampling.

What do we use for the autoregressive model in equation (11)? In our experiments, we follow [13] and define

$$\log p_\theta(\omega_t \mid \boldsymbol{\omega}_{<t}) \propto \exp(W\,[1; \mathbf{h}_{t-1}]) \tag{15}$$

for all $\omega_t \in \Omega \cup \{\textsc{eos}\}$, where $\mathbf{h}_{t-1}$ is the state of an LSTM after reading $\boldsymbol{\omega}_{<t}$.

D Hyperparameters

In this section, we describe the hyperparameters used for the frozen scorer and the various samplers in the experimental setup of §4.

For the frozen scorer $\tilde{p}_\theta$, we parameterize it with a two-layer LSTM with hidden dimension 256.

For samplers q_ϕ, the recurrent networks are one-layer. Though the model architecture varies (as described in §3), they all use the same hidden dimension size d for each of their recurrent networks (except that the bidirectional encoder for SWA uses hidden dimension $d/2$ in each direction, so that the dot product in equation (3) is conformable). Learned embeddings for the input, output, and mark symbols also all have dimension d.

The result presented in Fig. 4a is based on $d = 256$. Note that while the models are matched in dimensionality, they are unfortunately not matched in the number of parameters. We also vary the dimension size in Fig. 6 to see the effect of model capacity on the samplers. The other hyperparameters used consistently for all model training are shown in Table 3. As shown in the table, we use a small batch size (16); this is because each example in the batch samples $k = 16$ proposals, leading to $16 \times 16 = 256$ mark strings to generate or score in parallel.

We also employ common techniques like length penalty for very short sequences, as well as label smoothing, gradient clipping, etc.

Hyperparameter	Value
batch_size	16
k (# proposals)	16
K (for IWAE)	32
dropout	0.3
grad_clip	5.0
learning_rate	1×10^{-5}
tilde_p_lr	1×10^{-3}
label_smoothing	0.1
length_threshold	100
length_penalty	1.0

Table 3: Hyperparameter List

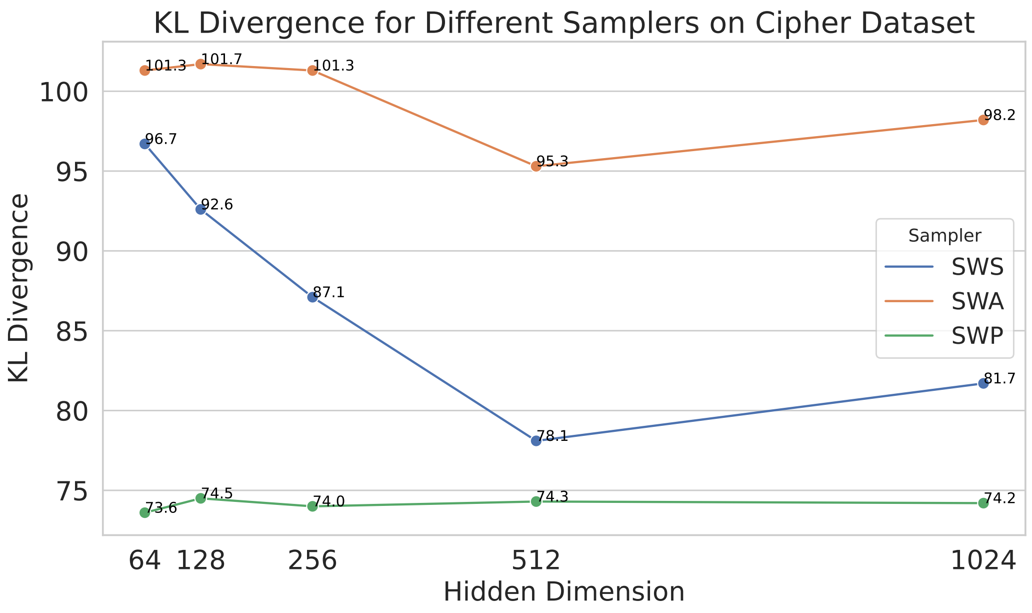

Figure 6: KL divergence for different samplers on `cipher` dataset, using base models with hidden dimensions varying from 64 to 1024.

E Qualitative Analysis

In Tables 4 to 6, we compare the mark sequences proposal by different samplers for each dataset to give a more thorough understanding of the proposal samplers' behavior. For `tr`, the input is "chalnay" and the output in Urdu is "<ur> 35 8 10 22" where <ur> is a fixed language code and the numbers denote Urdu symbols. As `tr` alignment is monotonic, we see that all three samplers give good alignments, where SWS and SWA propose the exact same path and SWP is slightly different in that **it made a mistake by aligning "c" to the language label "<ur>"**. For SWS and SWA, they align "c" to "35", "hal" to "8", "n" to "10", and "ay" to "22". For SWP, it aligns "c" to "<ur>", "hal" to "35 8", "n" to "10", and "ay" to "22". Therefore SWP is slightly wrong compared to the other two, which is also reflected in the average KL divergence in the main results (Fig. 4a).

`scan` is a hard task for all three samplers as the alignment is not necessarily monotonic (although the example we show here is actually monotonic). We find that SWS and SWA samplers produce somewhat interpretable results (which presumably correspond to high-scoring paths under p_θ), as they mix the generation of the input mark strings with the output strings. For example, SWS proposes "look around" followed by a sequence of actions, then proposes "jump around" that is followed by "I_Jump", etc. However, for SWP sampler, its path suggests that it did not know how to align these input and output marks because it simply dumps all the output marks first and then deletes all the input marks. As discussed in §5, we believe this is essentially a failure of training a complex model: SWP does not learn how to usefully embed the graph of monotonic suffix alignments corresponding to paths in the deletion/insertion FST.

Lastly, for `cipher`, the SWS and SWP samplers have proposed to use cipher2, which in this case allows high-probability alignments with many copy marks that explain output symbols as enciphered versions of input symbols. Given the choice of cipher2, SWP continues by finding a better alignment than SWS does. (SWS erred by enciphering the second "d" and the final "o" too early ("d→f", "o→i"), forcing it to delete those output characters and insert them again later.) The SWA proposal has unwisely proposed cipher3, resulting in a much longer and lower-probability alignment that does not explain any of the output symbols as enciphered input symbols, but must choose to insert all of them.

Input	chalnay
Output	<ur> 35 8 10 22
SWA	insert <ur>, delete c, insert 35, delete h, delete a, delete l, insert 8, delete n, insert 10, delete a, delete y, insert 22
SWS	same as SWA
SWP	delete c, insert <ur>, delete h, delete a, delete l, insert 35, insert 8, delete n, insert 10, delete a, delete y, insert 22

Table 4: Proposed mark strings given an $(\boldsymbol{x}, \boldsymbol{y})$ pair from the `tr` dataset.

Input	look around left twice and jump around right
Output	I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP
SWA	look I_TURN_LEFT around I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK left I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK twice I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_RIGHT I_JUMP and jump around I_TURN_RIGHT I_JUMP right I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP
SWS	look around I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT left I_LOOK I_TURN_LEFT twice I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK and I_TURN_RIGHT jump around I_JUMP I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP right I_TURN_RIGHT I_JUMP
SWP	I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK I_TURN_LEFT I_LOOK look I_TURN_LEFT I_LOOK I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP I_TURN_RIGHT I_JUMP around left twice and jump around

Table 5: Proposed mark strings given an $(\boldsymbol{x}, \boldsymbol{y})$ pair from the `scan` dataset.

Input	d d o r g r r i q a v o
Output	f j f i a e a a j k s z w i e
SWA	cipher3: d→m→ε, insert f, insert j, d→m→ε, insert f, insert i, o→x→ε, insert a, insert e, r→f→ε, insert a, insert a, g→z→ε, insert j, insert k, r→f→ε, insert s, insert z, r→f→ε, insert w, insert i, i→j→ε, insert e, q→a→ε, a→e→ε, v→h→ε, o→x→ε
SWS	cipher2, d→f, d→f→ε, insert j, insert f, o→i, r→a, g→e, r→a, r→a, i→j, q→k, a→s, v→z, o→i→ε, insert w, insert i, insert e
SWP	cipher2: d→f, insert j, d→f, o→i, r→a, g→e, r→a, r→a, i→j, q→k, a→s, v→z, insert w, o→i, insert e

Table 6: Proposed mark strings given an $(\boldsymbol{x}, \boldsymbol{y})$ pair from the `cipher` dataset. The notation "a→b" is actually an abbreviation for a sequence of 5 marks: replace, a, b, copy, b. The notation "a→b→ε" is also an abbreviation for a sequence of 5 marks: replace, a, b, delete, b.

F Baseline Sampler with No Lookahead

We trained a baseline sampler that does not utilize information from the future (suffix of input $\boldsymbol{x}$ and output $\boldsymbol{y}$). The parameterization is shown below:

$$q_\phi(\omega_t \mid \boldsymbol{\omega}_{<t}, \mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}) = \text{softmax}(W[1; \mathbf{h}_{t-1}]) \tag{16}$$

Task	TR	Scan	Cipher
SWA	1.0	7.9	1.1
SWS	1.1	4.3	1.3
SWP	1.0	1.1	1.0

Table 7: Deduplicated effective sample size (ESS) from different samplers across tasks. Instead of using the regular ESS, we first aggregate the probability mass for samples that are identical as we realize that many duplicated samples are proposed. Then we compute the normalized probability $\hat{w}$ over the deduplicated probabilities and compute the ESS $= \frac{1}{\hat{w}^2} = \frac{(\sum_{i=1}^{n} w_i)^2}{\sum_{i=1}^{n} w_i^2}$. We see that ESS is very small for `tr` and `cipher` task, showing that the proposal distribution is sharp after training.

We would expect the samplers with lookahead to give better performance compared to this no-lookahead baseline, but our result in Fig. 7 shows that SWP performs worse compared to it on `tr` and `scan` tasks. This suggests that SWP might suffer from training issues or that even a well-trained model with the SWP architecture tends to generalize poorly.

Sampler	TR	Scan	Cipher	Avg
SWA	23.3	38.1	101.3	54.3
SWS	**21.8**	**25.4**	87.1	**44.7**
SWP	38.1	102.9	**73.9**	71.7
No Lookahead	31.9	39.9	95.5	55.8

Figure 7: *Partial KL* divergence comparison (lower values are better) for different proposal samplers.

G Limitations

Inference Speed In this work, we mainly focus on the quality aspect of inference networks, rather than their speed. However, many applications of inference networks depend on their speed [12]; otherwise, comparable quality might be achieved using a more naive sampling strategy. For our samplers, SWA is the fastest because it directly computes attention over the representation of the raw strings. SWS is only slightly slower. Our novel method, SWP, is the slowest (about 10x time of SWA) as it propagates information over the large graph of possible alignments defined by the FST topology.

Cyclic MFSTs Our SWP sampler assumes that the MFST $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is acyclic. Since $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ is deterministic and hence ε-free with respect to the mark tape, this is equivalent to saying that for each $\boldsymbol{x}, \boldsymbol{y}$, the length of a mark string is bounded. This assumption is ordinarily satisfied, but might be violated if, for example, $\mathcal{T}$ were obtained by composing an MFST that can insert arbitrarily many symbols with another MFST that can delete them again. This would lead to infinitely many generating paths that transform $\boldsymbol{x}$ into $\boldsymbol{y}$, which would require a cyclic $\mathcal{T}_{\boldsymbol{x},\boldsymbol{y}}$ encode.

Larger Datasets and Models Due to the computation cost of the parameter estimation that requires sampling and state tracking, we are using three datasets with small vocabulary to showcase the difference between samplers. We will leave scaling up the dataset/model for future investigation.

Alternative Architectures We used recurrent neural networks as our main building block. Switching to Transformers, for example, would require a redesign of the SWP sampler to attend over arcs in the suffix graph. We also limited our experiments to single-layer recurrent networks.

Particle Smoothing Due to computation cost, we also did not perform full particle smoothing (used in [13]) because it requires sampling of multiple paths $\boldsymbol{z}$ in parallel and weighting the prefixes $\boldsymbol{z}_{<t}$ *at each time step* t by an estimate of their conditional probability. The benefit of calculating these weights at each stage lies in their ability to measure effective sample size (ESS) and facilitate multinomial resampling when the ESS drops below a certain threshold. As shown in Table 7, ESS indeed falls low as many duplicated paths are proposed.