Learnability on the Information-Theoretic Continuum: Inductive Bias for Information Locality in Neural Language Models

Anonymous ACL submission

Abstract

Inductive biases are inherent in every machine learning system, shaping how models generalize from finite data. In the case of neural language models (LMs), debates persist as to whether these biases align with or diverge from human processing constraints. To address this issue, we propose a quantitative framework that allows for controlled investigations into the nature of these biases. Within our framework, we introduce m-local entropy—an information-theoretic measure derived from average lossy-context surprisal—that captures the local uncertainty of a language by quantifying how effectively the $m-1$ preceding symbols disambiguate the next symbol. In experiments on both perturbed natural language corpora and languages defined by probabilistic finite-state automata (PFSA), we show that languages with higher m-local entropy are more difficult for Transformer and LSTM LMs to learn. These results suggest that neural LMs, much like humans, are highly sensitive to the local statistical structure of a language.

1 Introduction

Every machine learning system has some form of inductive bias; given a finite sample of data with infinitely many plausible generalizations, a system inherently prefers certain generalizations over others (Mitchell, 1980; Rawski and Heinz, 2019). This concept is central to the growing discussion of whether the inductive biases of neural network language models align with the cognitive pressures that shape human language learning. In a 2023 New York Times article, Chomsky et al. famously argued that neural language models possess inductive biases fundamentally different from human cognitive constraints, a claim that has motivated theoretical rebuttals (Piantadosi, 2024) as well as empirical research to test the extent of this divergence (Kallini et al., 2024; Ahuja et al., 2024). In particular, Kallini et al. (2024) demonstrated that

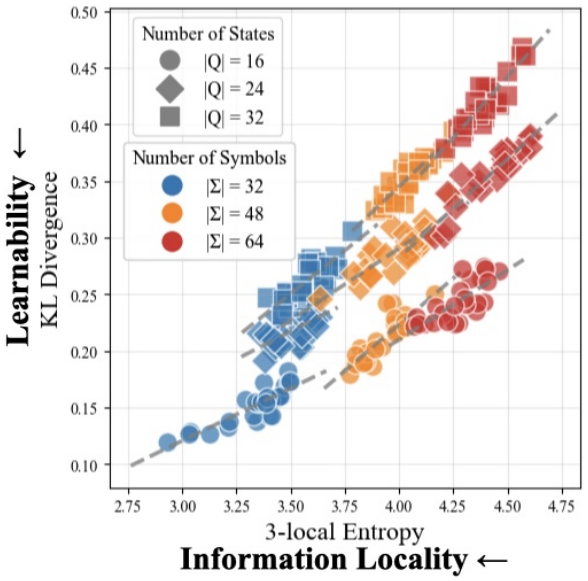

Figure 1: KL divergence (Transformer LM) as a function of the 3-local entropy of the language generated from a PFSA in Experiment 2. LMs perform better at languages with lower local entropy.

perturbing natural language corpora to alter their sequential structure—moving them along an intuitive continuum of impossibility—makes these languages harder for neural LMs to learn. While their findings suggest that disrupting local structure (e.g., through local shuffling transformations) impacts learnability, they do not isolate the specific linguistic properties responsible for this effect. To rigorously assess whether a language model's inductive biases align with human constraints, we must identify *quantifiable properties* of language that affect human learning difficulty and *systematically manipulate* these properties in controlled experiments with neural LMs. Information-theoretic models of language processing, which view the structure of languages as shaped by language users' joint optimization of informativity and complexity, provide a promising framework for identifying these properties.

In this paper, we resort to the principles of **information locality**, which suggest that language is structured to minimize linear distance between linguistic elements with high mutual information (Gibson, 1998, 2001; Futrell, 2019; Futrell et al., 2020). Information locality is thought to arise from the memory limitations of human processors, which make it challenging to integrate long-range linguistic dependencies (Hahn et al., 2022). The influence of these cognitive constraints has been observed across multiple timescales, in both language comprehension and production, and across diverse languages of the world (Hahn et al., 2021; Futrell, 2023; Futrell and Hahn, 2024). If we could demonstrate that a neural language model's learnability of a language is influenced by its local predictability, we would reveal an inductive bias that aligns with the functional pressures shaping human language processing.

As a first step in this direction, we propose using m-local entropy, an information-theoretic measure designed to quantify a linear notion of local predictability which can be derived from the principles of information locality (Futrell et al., 2020).[1] We study how m-local entropy affects learnability in two sets of experiments: one where we perturb natural language corpora, and another where we randomly generate PFSAs. We train LSTM and Transformer language models on these languages and examine whether language models' difficulty in learning a language is predicted by the language's m-local entropy, to see if language models and humans share an inductive bias for information locality. Our experiments demonstrate that m-local entropy negatively correlates with the ability of a language model to learn a language. Specifically, our experiment with an English natural language corpus—and its perturbed variants—reveals that both LSTM and Transformer architectures show systematic degradation in performance as m-local entropy increases, even when global entropy remains constant. Furthermore, in experiments using PFSAs, we manipulate the properties of languages more *systematically* and show that this trend is not an artifact of the corpora or particular perturbation functions used in our experiments.

[1] More specifically, m-local entropy can be derived from the lossy-context surprisal theory of language comprehension (Futrell et al., 2020), a theory belonging to the expectation-based family of theories of language processing (Hale, 2001; Levy, 2008). See §2.2.2 for details.

2 Formal Background

2.1 Languages and Language Models

An **alphabet** Σ is a finite, non-empty set of symbols. The **Kleene closure** Σ^* is the set of all strings with symbols from Σ. We use ε to denote the empty string, and $|\boldsymbol{y}|$ to denote the length of $\boldsymbol{y} \in \Sigma^*$. A **language** $\mathbb{L}$ is a subset of Σ^*.

A **language model** p is a probability distribution over Σ^*. The **prefix probability** $\overrightarrow{p}(\boldsymbol{y})$ is the probability that a string begins with $\boldsymbol{y} \in \Sigma^*$:

$$\overrightarrow{p}(\boldsymbol{y}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y}' \in \Sigma^*} p(\boldsymbol{y}\boldsymbol{y}') \tag{1}$$

Given prefix probabilities, the conditional probability of the continuation $\boldsymbol{y}' \in \Sigma^*$ given a preceding context $\boldsymbol{y}$ can be computed as

$$p(\boldsymbol{y}' \mid \boldsymbol{y}) = \frac{\overrightarrow{p}(\boldsymbol{y}\boldsymbol{y}')}{\overrightarrow{p}(\boldsymbol{y})}. \tag{2}$$

With this, we can factorize a language model p as

$$p(\boldsymbol{y}) = p(\text{EOS} \mid \boldsymbol{y}) \prod_{t=1}^{|\boldsymbol{y}|} p(y_t \mid \boldsymbol{y}_{<t}), \tag{3}$$

where each $p(y_t \mid \boldsymbol{y}_{<t})$ is a distribution over $\overline{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\text{EOS}\}$, where $\text{EOS} \notin \Sigma$ is a distinguished end-of-string symbol, and

$$p(\text{EOS} \mid \boldsymbol{y}) \stackrel{\text{def}}{=} \frac{p(\boldsymbol{y})}{\overrightarrow{p}(\boldsymbol{y})}. \tag{4}$$

We define p's **infix probability** $\overleftrightarrow{p}$ as

$$\overleftrightarrow{p}(\boldsymbol{y}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y}' \in \Sigma^*} \sum_{\boldsymbol{y}'' \in \Sigma^*} p(\boldsymbol{y}'\boldsymbol{y}\boldsymbol{y}''). \tag{5}$$

Note that, despite denoting probabilities of events, $\overrightarrow{p}$ and $\overleftrightarrow{p}$ are *not* probability distributions. In general, the sums $\overrightarrow{Z} \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} \overrightarrow{p}(\boldsymbol{y})$ and $\overleftrightarrow{Z} \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} \overleftrightarrow{p}(\boldsymbol{y})$ may diverge. A sufficient condition for the former to be finite is that the expected length $\mu \stackrel{\text{def}}{=} \mathbb{E}_{\boldsymbol{y} \sim p} |\boldsymbol{y}|$ under p is finite, which follows from the relationship $\overrightarrow{Z} = \mu + 1$.[2] In the following, we assume that $\overrightarrow{Z} < \infty$, which means that we will be able to *normalize* the prefix probabilities to arrive at a probability distribution over string prefixes. We do not put this restriction on $\overleftrightarrow{Z}$ since we will normalize $\overleftrightarrow{p}$ over a finite subset of Σ^*.

[2] The proof is similar to Borenstein et al. (2024, Lem. D.1).

2.2 Global Entropy and M-local Entropy

The concept of *entropy*, as introduced by Shannon (1948), provides a foundational framework for quantifying uncertainty in language. Depending on how one defines the underlying probability distribution over linguistic units, entropy can capture different aspects of language complexity. In this paper, we derive two versions of entropy—*global* entropy (i.e., the Shannon entropy) and *m-local* entropy—each capturing different facets of language complexity.

2.2.1 Global Entropy

The **global entropy** of p is defined as

$$\mathrm{H}(p) \stackrel{\text{def}}{=} -\sum_{\boldsymbol{y}\in\Sigma^*} p(\boldsymbol{y}) \log p(\boldsymbol{y})\,. \tag{6}$$

This definition reflects the uncertainty in the distribution over all possible strings $\boldsymbol{y} \in \Sigma^*$: Higher global entropy indicates that p distributes probability mass more uniformly across strings. We further define the (global) next-symbol entropy for finite-mean-length language models. It is a weighted average of the entropy of all local next-symbol distributions, averaging over all possible contexts $\boldsymbol{y} \in \Sigma^*$, and weighted by the normalized prefix probability of $\boldsymbol{y}$.[3] For that, define the $\overline{\Sigma}$-valued random variable $Y_{\boldsymbol{y}}$ distributed as

$$p\left(Y_{\boldsymbol{y}} = y\right) = p(y \mid \boldsymbol{y})\,. \tag{7}$$

We have $\mathrm{H}(Y_{\boldsymbol{y}}) \stackrel{\text{def}}{=} -\sum_{y\in\overline{\Sigma}} p(y \mid \boldsymbol{y}) \log p(y \mid \boldsymbol{y})$. Then, we define the **next-symbol** entropy as

$$\mathrm{H}_\Sigma(p) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y}\in\Sigma^*} \frac{\overrightarrow{p}(\boldsymbol{y})}{\overrightarrow{Z}} \mathrm{H}(Y_{\boldsymbol{y}}) \tag{8a}$$

$$= \frac{1}{\mu+1} \sum_{\boldsymbol{y}\in\Sigma^*} \overrightarrow{p}(\boldsymbol{y}) \mathrm{H}(Y_{\boldsymbol{y}}). \tag{8b}$$

The following fact is a special case of Malagutti et al. (2024, Thm. 2.2).

Lemma 2.1. *Let p be a language model with $\mu < \infty$. Then, we have*

$$\mathrm{H}_\Sigma(p) = \frac{\mathrm{H}(p)}{\mu+1}. \tag{9}$$

Invariance of global and next-symbol entropy. Global entropy is insensitive to bijective transformations of Σ^*. This has important implications for using entropy as a predictor of language learning difficulty. For example, global entropy does not change under string permutations and thus does *not* account for local ambiguity or variations in predictability within different segments of a string. The same holds for next-symbol entropy under bijective *length-preserving* transformations due to Lemma 2.1. We investigate this further in §3, but first we treat it more formally. Let p be a language model and p' be a *perturbed* language model where $f\colon \Sigma^* \to \Sigma^*$ is a bijection and $p'(\boldsymbol{y}) = p\big(f^{-1}(\boldsymbol{y})\big)$ for every $\boldsymbol{y} \in \Sigma^*$. Crucially, p and p' have the same global entropy: The global entropy of p' is

$$\mathrm{H}(p') = -\sum_{\boldsymbol{y}'\in\Sigma^*} p\big(f^{-1}(\boldsymbol{y}')\big)\, \log p\big(f^{-1}(\boldsymbol{y}')\big). \tag{10}$$

Since f is a bijection, we can reindex the sum by letting $\boldsymbol{y} = f^{-1}(\boldsymbol{y}')$; as $\boldsymbol{y}'$ ranges over all of Σ^*, so does $\boldsymbol{y}$. Hence,

$$\mathrm{H}(p') = -\sum_{\boldsymbol{y}\in\Sigma^*} p(\boldsymbol{y}) \log p(\boldsymbol{y}) = \mathrm{H}(p). \tag{11}$$

Furthermore, if the bijection also preserves string length, it follows from Lemma 2.1 and Eq. (11) that next-symbol entropy is preserved as well.

2.2.2 M-local Entropy

Global next-symbol entropy measures uncertainty over next-symbol predictions conditioned on the full available context, averaged across all possible contexts. This can be seen as the limit of a *local* quantitification of uncertainty, which captures the unpredictability of next-symbol predictions given a fixed amount of preceding context. We term this fixed-context uncertainty measure **local entropy**.

Let $\boldsymbol{C}$ be a Σ^{m-1}-valued random variable distributed according to $\overleftrightarrow{p}$ normalized over Σ^{m-1}:

$$p\left(\boldsymbol{C} = \boldsymbol{c}\right) \stackrel{\text{def}}{=} \frac{\overleftrightarrow{p}(\boldsymbol{c})}{\sum_{\boldsymbol{c}'\in\Sigma^{m-1}} \overleftrightarrow{p}(\boldsymbol{c}')}. \tag{12}$$

$p\left(\boldsymbol{C} = \boldsymbol{c}\right)$ can be interpreted as observing $\boldsymbol{c}$ as a length-$(m-1)$ substring of a string from p. Let $Y_{\boldsymbol{c}}$ be the $\overline{\Sigma}$-valued random variable distributed as

$$p\left(Y_{\boldsymbol{c}} = y_t\right) = p(y_t \mid \boldsymbol{y}_{t-m+1:t-1} = \boldsymbol{c})\,, \tag{13}$$

i.e., as the next symbol given that the previous $m-1$ symbols were $\boldsymbol{c}$. With slight abuse of notation, we write $p(y_t \mid \boldsymbol{c})$ to mean $p(y_t \mid \boldsymbol{y}_{t-m+1:t-1} = \boldsymbol{c})$.

[3] The weighting cannot be uniform, as Σ^* is infinite.

The use of $\boldsymbol{c}$ for $(m-1)$-length contexts will disambiguate this from conditioning on a general prefix $\boldsymbol{y} \in \Sigma^*$. Given any $\boldsymbol{c} \in \Sigma^{m-1}$, we can compute

$$\mathrm{H}(Y_{\boldsymbol{c}}) \stackrel{\text{def}}{=} -\sum_{y \in \overline{\Sigma}} p\left(y \mid \boldsymbol{c}\right) \log p\left(y \mid \boldsymbol{c}\right). \quad (14)$$

This captures the unpredictability of a symbol y after observing a given local context $\boldsymbol{c}$. We can then define the ***m*-local entropy** of p as an expectation over possible contexts $\boldsymbol{c} \in \Sigma^{m-1}$, with each context weighted by $p\left(\boldsymbol{C} = \boldsymbol{c}\right)$:

$$\begin{aligned} \mathrm{H}_m\left(p\right) &\stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\boldsymbol{c} \sim p(\boldsymbol{C}=\boldsymbol{c})}\left[\mathrm{H}(Y_{\boldsymbol{c}})\right] \quad (15) \\ &= \sum_{\boldsymbol{c} \in \Sigma^{m-1}} p\left(\boldsymbol{C} = \boldsymbol{c}\right) \mathrm{H}(Y_{\boldsymbol{c}}). \end{aligned}$$

This yields a measure of local complexity that can differ from global entropy. Even when two languages have identical *global* entropy, their *m-local* entropies can differ, reflecting differences in how reliably the recent context predicts the next symbol. Importantly, unlike global entropy, local entropy is *not* necessarily preserved under bijective transformations of Σ^*, which enables us to assess the impact of such transformations on learnability. As we show later, transformations that alter local structure can significantly influence how well neural LMs learn a language.

***M*-local entropy and lossy-context surprisal.** As a generalization of the surprisal model of language processing difficulty (Hale, 2001; Levy, 2008), Futrell et al. (2020) propose *lossy-context surprisal*. In this model, the predicted difficulty for processing an upcoming word y_t is a function of the word's expected log probability given a *lossy* memory representation r of the preceding context $\boldsymbol{c}_{<t}$:

$$\text{Difficulty}\left(y_t; \boldsymbol{c}_{<t}\right) \propto \mathop{\mathbb{E}}_{r \sim M(\boldsymbol{c}_{<t})}\left[-\log p\left(y_t \mid r\right)\right], \quad (16)$$

where M is a memory encoding function which gives us the conditional distribution of a memory representation r given the previous context $\boldsymbol{c}_{<t}$. If we assume that M always retains only the $m-1$ symbols immediately preceding y_t, then $r = M(\boldsymbol{c}_{<t}) = y_{t-m+1} \cdots y_{t-1}$, and surprisal becomes $-\log p\left(y_t \mid y_{t-m+1} \cdots y_{t-1}\right)$. The expectation of this surprisal over all possible contexts and next symbols is a special case of the *average (lossy-context) surprisal* (Futrell, 2019; Hahn et al., 2021) of a language, which corresponds to our definition of m-local entropy in Eq. (15). To our knowledge, no prior work has linked lossy-context surprisal directly to language model learnability—a connection that our work aims to explore.

2.3 Probabilistic Finite-state Automata

Definition 2.1. *A **probabilistic finite-state automaton** (PFSA) is a 5-tuple $(\Sigma, Q, \delta, \lambda, \rho)$ where*

- *Σ is an alphabet,*
- *Q is a finite set of states,*
- *$\delta \subseteq Q \times \Sigma \times [0,1] \times Q$ is a finite set of weighted transitions rendered as $q \xrightarrow{y/w} q'$,*
- *$\lambda, \rho\colon Q \to [0,1]$ are the initial and final weighting functions,*
- *λ satisfies $\sum_{q \in Q} \lambda\left(q\right) = 1$, and*
- *for all $q \in Q$, $\sum_{q \xrightarrow{y/w} q' \in \delta} w + \rho\left(q\right) = 1$.*

A **path** $\boldsymbol{\pi}$ in a PFSA $\mathcal{A}$ is a sequence of consecutive transitions $q_0 \xrightarrow{y_1/w_1} \cdots \xrightarrow{y_N/w_N} q_N$. We define its **scan** as $\mathbf{s}\left(\boldsymbol{\pi}\right) \stackrel{\text{def}}{=} y_1 \cdots y_N$. $\Pi(\mathcal{A}, \boldsymbol{y})$ denotes the set of all paths in $\mathcal{A}$ that scan $\boldsymbol{y} \in \Sigma^*$. The **inner path weight** of $\boldsymbol{\pi}$ is $\overline{\boldsymbol{w}}(\boldsymbol{\pi}) = \prod_{n=1}^{N} w_n$ and its **path weight** is $\boldsymbol{w}(\boldsymbol{\pi}) = \lambda(q_0)\overline{\boldsymbol{w}}(\boldsymbol{\pi})\rho(q_N)$.

A PFSA $\mathcal{A}$ induces a language model $p_{\mathcal{A}}$ as

$$p_{\mathcal{A}}(\boldsymbol{y}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y})} \boldsymbol{w}(\boldsymbol{\pi}). \quad (17)$$

Studying PFSAs not only allows us to perform controlled experiments but also enables us to compute many quantities of interest exactly. App. A contains a collection of closed-form solutions for computing various quantities of interest, including the string (prefix and infix) probabilities and the m-local entropy of the induced language model.

3 Experiment 1: LM Performance along the *M*-local Entropy Continuum

In the first experiment, we investigate the relationship between local entropy and LM performance using a natural language corpus. We hypothesize that *local entropy* is a key factor determining how easily an LM learns a language. To test this hypothesis, we apply a bijective perturbation function to a natural language corpus that alters its local structure. This results in a counterfactual perturbed corpus (cf. Kallini et al., 2024), which has different local entropy from the original one but the same global entropy. We then train LMs on the naturally occurring corpus and the perturbed one and study how local entropy affects the LMs' performance.

3.1 Constructing Languages with Different Local Complexity

Here, we detail several specific transformations implemented in our experiments, refining the perturbation functions of Kallini et al. (2024).

DETERMINISTICSHUFFLE. A fixed random permutation σ of the string positions $\{1, \ldots, T\}$ is applied to any string of length T. This permutation is *deterministic* throughout the experiment, ensuring consistent shuffling whenever it is used.

REVERSE. This function reverses the entire sequence of symbols. Formally, given a string $\boldsymbol{y} = y_1\, y_2\, \ldots\, y_T$, the REVERSE mapping produces $y_T\, y_{T-1}\, \ldots\, y_1$. It is trivially invertible (by applying the same operation again), making it a bijection.

EVENODDSHUFFLE/ODDEVENSHUFFLE. Let $\boldsymbol{y} = y_1 y_2 \ldots y_T$ be a string of length T. Define two subsequences $O(\boldsymbol{y}) = y_1 y_3 y_5 \ldots$ and $E(\boldsymbol{y}) = y_2 y_4 y_6 \ldots$ that collect all symbols in $\boldsymbol{y}$ at *even* positions and *odd* positions, respectively. We then define EVENODDSHUFFLE$(\boldsymbol{y}) = E(\boldsymbol{y})O(\boldsymbol{y})$ and ODDEVENSHUFFLE$(\boldsymbol{y}) = O(\boldsymbol{y})E(\boldsymbol{y})$.

K-LOCALDETERMINISTICSHUFFLE. Let $\boldsymbol{y} = y_1\, y_2\, \ldots\, y_T$ be a string in Σ^*, which we partition into consecutive windows of size k. For the i-th window, $y_{(i-1)k+1}, \ldots, y_{ik}$, we apply a fixed permutation π_i determined by i and a global random seed.[4] Formally, the K-LOCALDETERMINISTICSHUFFLE of $\boldsymbol{y}$ produces $\big(\pi_1(y_1, \ldots, y_k),\ \pi_2(y_{k+1}, \ldots, y_{2k}),\ \ldots\big)$.

3.2 Estimating the *M*-local Entropy of a Language

Unfortunately, the true m-local entropy is not directly accessible for these corpora since we don't know the underlying probability distribution of natural language. In this experiment, we estimate it using an n-gram language model implemented with KenLM (Heafield, 2011). Given a corpus $\mathbb{D}$, we train an n-gram model on $\mathbb{D}$ to get the estimated conditional probability distribution$\hat{p}(y \mid \boldsymbol{c})$ for $\boldsymbol{c} \in \Sigma^{m-1}$. Plugging this estimated probability distribution into Eq. (14), we can compute

$$\hat{\mathrm{H}}(Y_{\boldsymbol{c}}) = -\frac{1}{N(\boldsymbol{c})} \sum_{y \in \mathbb{D}} \log \hat{p}\,(y \mid \boldsymbol{c})\,, \tag{18}$$

where $N(\boldsymbol{c})$ is the number of times $\boldsymbol{c}$ appears in $\mathbb{D}$. The normalized infix probability is estimated as

$$\hat{p}(\boldsymbol{C} = \boldsymbol{c}) = \frac{N(\boldsymbol{c})}{N_{total}}, \tag{19}$$

where $N_{total} = \sum_{\boldsymbol{c}' \in \Sigma^{m-1}} N(\boldsymbol{c}')$.

Given these and Eq. (15), we can compute estimated m-local entropy as

$$\hat{\mathrm{H}}_m(p) = \sum_{\boldsymbol{c} \in \Sigma^{m-1}} \hat{p}(\boldsymbol{C} = \boldsymbol{c})\ \hat{\mathrm{H}}(Y_{\boldsymbol{c}}) \tag{20a}$$

$$= -\frac{1}{N_{total}} \sum_{\boldsymbol{c}y \in \mathbb{D}} \log \hat{p}(y \mid \boldsymbol{c}) \tag{20b}$$

This estimator is a practical proxy for the quantity in Eq. (15). The m-local entropy of each corpus is estimated by an n-gram model with order $m-1$ trained on the concatenation of the training, validation, and test set of the corpus.

3.3 Experimental Setup

3.3.1 Neural Language Models

We investigate how varying *local entropy* in a language impacts the performance of two widely used neural LM architectures: the LSTM (Gers and Schmidhuber, 2001) and a causally-masked Transformer encoder (Vaswani et al., 2017). We use a single-layer LSTM with 512-dimensional hidden units and a 4-layer causally-masked Transformer with 768-dimensional embeddings, 3072-dimensional feedforward layers, and 12 attention heads. Both are implemented in PyTorch (Paszke et al., 2019). Both architectures are trained on the training set via the standard language modeling objective across 5 random training seeds. See Appendices C and D for more details.

3.3.2 Dataset

We conduct our experiments on a subset of the Brown Laboratory for Linguistic Information Processing 1987–89 Corpus Release 1 (BLLIP; Charniak et al., 2000).[5] Specifically, we adopt the same training, development, and test splits as BLLIP-SM in Hu et al. (2020), which comprise roughly 200K sentences, totaling around 5M tokens. Starting from this original corpus, we apply the perturbation functions in §3.1 to produce perturbed corpora. For both DETERMINISTICSHUFFLE and K-LOCALDETERMINISTICSHUFFLE, we

[4] If the string length T is not a multiple of k, then the final window, which contains fewer than k symbols, is permuted by applying the fixed permutation to all the available symbols in that window.

[5] We also conduct the same set of experiments using the BabyLM corpus (Choshen et al., 2024); results in App. E.2.

	2-local entropy	3-local entropy	4-local entropy	5-local entropy
BASE	6.67	4.27	2.92	2.45
REVERSE	6.98	4.39	2.98	2.51
EVENODDSHUFFLE	7.91	5.10	3.76	3.43
ODDEVENSHUFFLE	7.87	5.07	3.74	3.41
LOCALSHUFFLE (K=3)	8.12 ± 0.08	5.05 ± 0.06	3.68 ± 0.07	3.39 ± 0.07
LOCALSHUFFLE (K=4)	8.25 ± 0.07	5.17 ± 0.04	3.80 ± 0.04	3.56 ± 0.05
LOCALSHUFFLE (K=5)	8.33 ± 0.07	5.25 ± 0.04	3.88 ± 0.04	3.64 ± 0.05
LOCALSHUFFLE (K=6)	8.43 ± 0.07	5.31 ± 0.05	3.97 ± 0.06	3.72 ± 0.06
LOCALSHUFFLE (K=7)	8.47 ± 0.07	5.36 ± 0.05	4.03 ± 0.06	3.78 ± 0.07
DETERMINISTICSHUFFLE	8.77	5.73	4.60	4.41

Table 1: *M*-local entropy values for the BASE (original) corpus and the different perturbed corpora. LOCAL SHUFFLE refers to the K-LOCALDETERMINISTICSHUFFLE. Values are shown as mean ± standard deviation (averaged over different random seeds).

use 20 random seeds. In the case of K-LOCALDETERMINISTICSHUFFLE, we vary the parameters k over the set $\{3, 4, 5, 6, 7\}$, yielding 20 perturbed corpora for each k. This produces a total of 124 distinct "languages," including other perturbed corpora and the BASE (original) corpus.

3.3.3 Evaluating Learning Difficulty

In this experiment, we rely on *next-symbol cross-entropy* as a measure of how well a trained language model q approximates the target distribution. If p is the ground-truth language model, and q is any learned neural LM, we can estimate the next-symbol cross-entropy as:

$$\hat{\mathrm{H}}_{\Sigma}(p, q) = \tag{21}$$
$$-\frac{1}{S}\sum_{\boldsymbol{y}\in\mathbb{D}}\Big[\log q\big(\text{EOS} \,\big|\, \boldsymbol{y}\big) + \sum_{t=1}^{|\boldsymbol{y}|}\log q\big(y_t \,\big|\, \boldsymbol{y}_{<t}\big)\Big],$$

where $\mathbb{D} = \{\boldsymbol{y}^{(n)}\}_{n=1}^{N}$ is a set of i.i.d. draws from p and $S = \sum_{\boldsymbol{y}\in\mathbb{D}} |\boldsymbol{y}| + 1$. In this experiment, we evaluate each LM using the estimated next-symbol cross-entropy on the test set. When comparing the "learnability" of two languages, it's important to take into account their inherent entropy since learning a language means learning its distribution, i.e., getting close to its lower bound entropy. Comparing absolute perplexity/cross-entropy can be misleading and could lead to different results. In fact, it is wrong to discuss the learnability of two languages with different inherent entropy using cross-entropy (e.g., NONDETERMINISTICSHUFFLE in Kallini et al., 2024). In this experiment, we can safely compare cross-entropy since all the corpora are assured to have the same inherent (global) entropy (see §3.1).

3.4 Results

How do different perturbations affect *m*-local entropy? Table 1 reports the m-local entropy values ($m \in \{2, 3, 4, 5\}$) for the BASE corpus and the various perturbed corpora. REVERSE barely changes the m-local entropy, whereas EVENODDSHUFFLE and ODDEVENSHUFFLE increase it somewhat more. In contrast, K-LOCALDETERMINISTICSHUFFLE yields progressively higher entropy as the window size k grows, indicating a greater disruption of local ordering. Finally, DETERMINISTICSHUFFLE produces the highest m-local entropies among all transformations.

These results confirm that the bijective transformations we defined in §3.1 effectively generate new languages with different *m-local* entropy than the original one, while preserving the *global* entropy by design. This yields a continuum of languages along a specific measurable axis of complexity rather than a qualitative notion of possibility as in Kallini et al. (2024).

How does *m*-local entropy affect LM performance? Figure 2 shows the relationship between the m-local entropy (estimated by m-gram models; §3.2) and the next-symbol cross-entropy of each neural LM on the test set. We observe a strong positive correlation between m-local entropy and next-symbol cross-entropy for both neural architectures. For example, with $m = 4$, the coefficient of determination R^2 reaches 0.922 for the LSTM LM

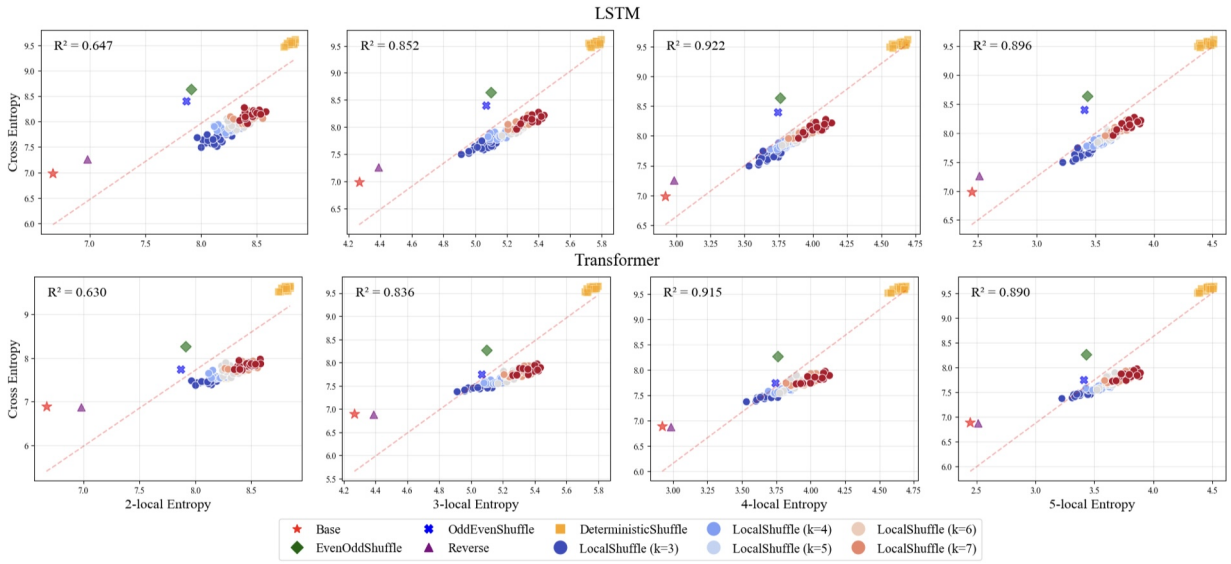

Figure 2: Scatter plots of next-symbol cross-entropy (y-axis) versus m-local entropy (x-axis) for $m \in \{2, 3, 4, 5\}$, for both LSTM (top row) and Transformer LM (bottom row). Each marker type/color corresponds to a different perturbation (e.g., `Reverse`, `DeterministicShuffle`, K-LOCALDETERMINISTICSHUFFLE with various window sizes, etc.). The red star indicates the unperturbed `Base` condition (original corpus). The dashed line in each panel is a linear fit, with R^2 indicating the coefficient of determination.

and 0.915 for the Transformer LM, indicating that higher local ambiguity (as measured by m-local entropy) generally leads to decreased performance (i.e., higher next-symbol cross-entropy) under both models. Furthermore, since our transformations are designed to preserve global entropy and global next-symbol entropy, these results highlight the crucial role of *local* entropy in the learnability of a language by neural LMs. This suggests that neural LMs inherently possess an inductive bias toward languages with lower local entropy.

4 Experiment 2: Controlled Learnability Tests with PFSAs

Experiment 1 only focused on a specific English corpus and a specific set of perturbation functions. To confirm that the results are not just an artifact of this experimental design, but a fundamental property of neural LMs, we conduct a controlled experiment using PFSAs. This also enables us to compute quantities of interest exactly, especially the m-local entropy of the induced language model.

4.1 Experimental Setup

We use the same LMs and training configurations as in §3, but we generate datasets using PFSAs (§4.1.1) and evaluate LMs while controlling for global entropy (§4.1.2).[6]

4.1.1 Generating Datasets using PFSAs

We construct random PFSAs with alphabet sizes $|\Sigma| \in \{32, 48, 64\}$ and numbers of states $|Q| \in \{16, 24, 32\}$. For each of the nine configurations, we randomly generate 25 automata. We control the randomness with five random seeds determining the PFSA topology (the underlying multi-graph) and five random seeds determining the transition weights. See Algorithm 1 in App. B for details of the generation. We sample 20K strings for the training set, 5K for the validation set, and 5K for the test set from $p_{\mathcal{A}}$ for each PFSA $\mathcal{A}$.

4.1.2 Evaluating Learning Difficulty

Using PFSAs allows us to compute a range of entropy-related values, including the inherent next-symbol entropy (§2.3), which enables us to evaluate LMs based on KL divergence D_{KL}. Specifically, the estimated $\hat{D}_{\mathrm{KL}}$ is given by subtracting the next-symbol entropy of the PFSA (Lemma 2.1) from the estimated next-symbol cross-entropy of the LM (Eq. (21)): $\hat{D}_{\mathrm{KL}} = \hat{\mathrm{H}}_{\Sigma}(p, q) - \hat{\mathrm{H}}_{\Sigma}(p)$. In this second experiment, we evaluate each LM using $\hat{D}_{\mathrm{KL}}$ on the test set.

[6] Recall that in Experiment 1 it was unnecessary to control for global entropy since it is preserved by design by the bijective transformations.

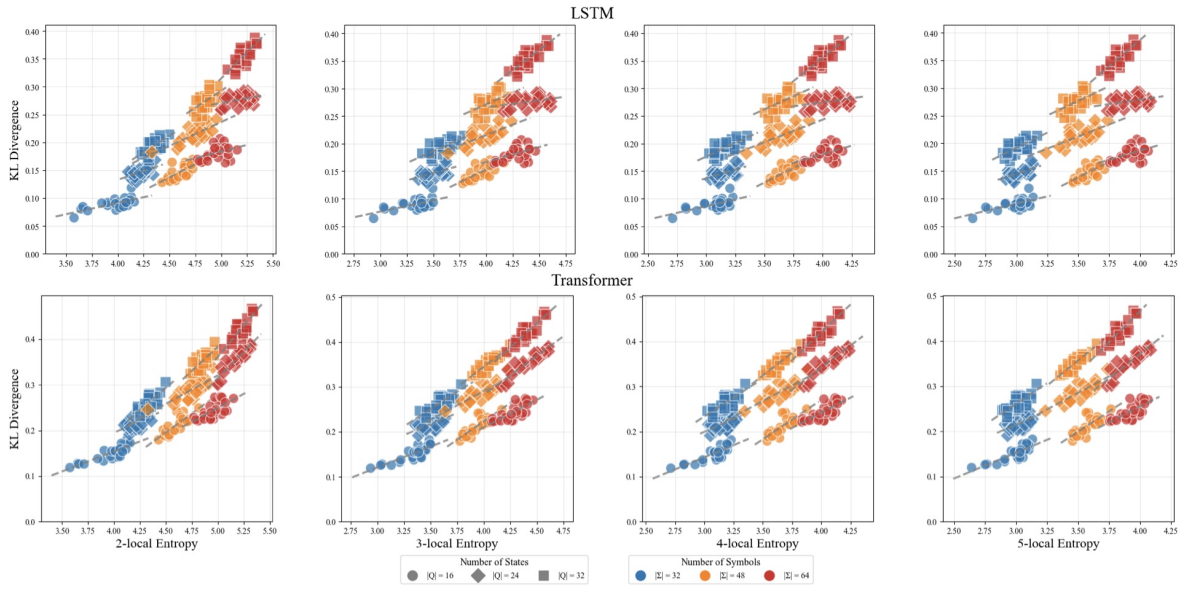

Figure 3: Scatter plots of symbol-level KL divergence (y-axis) versus m-local entropy (x-axis) for $m \in \{2, 3, 4, 5\}$, for both LSTM (top row) and causally-masked Transformer encoder (Transformer; bottom row) models. Each marker type/color corresponds to a different combination of number of states ($|Q|$) and symbols ($|\Sigma|$). The dashed line is a linear fit for each cluster.

4.2 Results: How Does M-local Entropy Affect LM Performance?

Figure 3 shows the relationship between the m-local entropy of PFSAs (calculated analytically; App. A) and the KL divergence of each neural LM on the test set; see Table 2 for Pearson correlation coefficients. The experimental results reveal a clear positive correlation between m-local entropy and $\hat{D}_{\mathrm{KL}}$ across both architectures and all values of $m = 2, 3, 4, 5$, indicating that neural language models find it more challenging to model distributions with higher local uncertainty. The Transformer LM consistently shows higher $\hat{D}_{\mathrm{KL}}$ compared to the LSTM within each topological cluster, suggesting that LSTMs are more effective at modeling these particular probability distributions (Weiss et al., 2018; Borenstein et al., 2024). Additionally, when $|\Sigma|$ is constant, $\hat{D}_{\mathrm{KL}}$ is higher for PFSAs with larger $|Q|$, which is consistent with the results of Borenstein et al. (2024).

5 Discussion and Conclusion

By proposing local m-local entropy as a predictor of learning difficulty grounded in lossy-context surprisal theory and information locality principles, we provide a formal information-theoretic perspective that connects the inductive biases of LMs and the statistical properties of language thought to be shaped by functional pressures in humans (Gibson, 2001; Futrell et al., 2020). Through two sets of experiments—one on perturbations of a natural language corpus and another using PFSAs for the controlled generation of test languages—we consistently find that both LSTM and Transformer architectures model languages with lower m-local entropy more effectively. The shared sensitivity to information locality between artificial and human learners suggests a common inductive bias shaping both systems, possibly because both systems process language incrementally.

Our findings open several promising directions for future research. One avenue is to explore inductive biases beyond information locality, such as sensitivity to hierarchical structure or structure dependence (Chomsky, 1957; Everaert et al., 2015), in order to better understand the full range of factors influencing language learnability in both humans and machines. Additionally, incorporating local entropy into model evaluation or as a regularization signal during training could lead to more robust and cognitively plausible language models (Timkey and Linzen, 2023; De Varda and Marelli, 2024).

In summary, our study presents new evidence of the strong sensitivity of neural language models to a language's local statistical structure, advancing our understanding of their inductive biases and establishing a foundation for future research on assessing and improving the alignment between artificial and human language processors.

Limitations

While our study reveals a strong correlation between m-local entropy and LM performance, it is important to note that our analysis remains correlational. We have not yet pinpointed the precise mechanisms by which variations in local uncertainty impact the learning dynamics of neural language models.

Additionally, our controlled experiments relied on PFSAs (PFSA) to generate languages with varied m-local entropy. Although PFSAs provide a tractable framework for such investigations, they capture only a limited set of the possible languages. It is plausible that employing more expressive formalisms—such as pushdown automata or even higher-level models—might reveal different relationships between local entropy and model performance. In fact, there are not a few empirical results that some types of neural language models don't necessarily seem to have similar inductive biases as humans (McCoy et al., 2020; Yedetore et al., 2023, *inter alia*).

Furthermore, our focus on information locality, as measured by m-local entropy, does not preclude the influence of other inductive biases that may also play significant roles in learning. Future work will need to disentangle these factors to fully understand their individual and combined effects on neural language models.

Ethical considerations

We employed AI-based tools (ChatGPT and GitHub Copilot) for writing and coding assistance. These tools were used in compliance with the ACL Policy on the Use of AI Writing Assistance.

References

Kabir Ahuja, Vidhisha Balachandran, Madhur Panwar, Tianxing He, Noah A. Smith, Navin Goyal, and Yulia Tsvetkov. 2024. Learning syntax without planting trees: Understanding when and why transformers generalize hierarchically. *ArXiv*, abs/2404.16367.

Nadav Borenstein, Anej Svete, Robin Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, and Ryan Cotterell. 2024. What languages are easy to language-model? A perspective from learning probabilistic regular languages. *Preprint*, arXiv:2406.04289.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. 2025. Training neural networks as recognizers of formal languages. In *The Thirteenth International Conference on Learning Representations*.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. BLLIP 1987-89 WSJ Corpus Release 1.

Noam Chomsky. 1957. *Syntactic structures*. Mouton.

Noam Chomsky, Ian Roberts, and Jeffrey Watumull. 2023. Noam chomsky: The false promise of ChatGPT. *The New York Times*.

Leshem Choshen, Ryan Cotterell, Michael Y. Hu, Tal Linzen, Aaron Mueller, Candace Ross, Alex Warstadt, Ethan Wilcox, Adina Williams, and Chengxu Zhuang. 2024. The 2nd BabyLM challenge: Sample-efficient pretraining on a developmentally plausible corpus. *Preprint*, arXiv:2404.06214.

Andrea De Varda and Marco Marelli. 2024. Locally biased transformers better align with human reading times. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pages 30–36, Bangkok, Thailand. Association for Computational Linguistics.

Martin B.H. Everaert, Marinus A.C. Huybregts, Noam Chomsky, Robert C. Berwick, and Johan J. Bolhuis. 2015. Structures, not strings: Linguistics as part of the cognitive sciences. *Trends in Cognitive Sciences*, 19(12):729–743.

Richard Futrell. 2019. Information-theoretic locality properties of natural language. In *Proceedings of the First Workshop on Quantitative Syntax (Quasy, SyntaxFest 2019)*, pages 2–15, Paris, France. Association for Computational Linguistics.

Richard Futrell. 2023. Information-theoretic principles in incremental language production. *Proceedings of the National Academy of Sciences of the United States of America*, 120.

Richard Futrell, Edward Gibson, and Roger P. Levy. 2020. Lossy-context surprisal: An information-theoretic model of memory effects in sentence processing. *Cognitive Science*, 44(3):e12814.

Richard Futrell and Michael Hahn. 2024. Linguistic structure from a bottleneck on sequential information processing. *ArXiv*, abs/2405.12109.

Felix A. Gers and Jürgen Schmidhuber. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Edward Gibson. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68(1):1–76.

Edward Gibson. 2001. The dependency locality theory: A distance-based theory of linguistic complexity. In *Image, Language, Brain: Papers from the First Mind Articulation Project Symposium*. The MIT Press.

Michael Hahn, Judith Degen, and Richard Futrell. 2021. Modeling word and morpheme order in natural language as an efficient trade-off of memory and surprisal. *Psychological Review*, 128(4):726–756.

Michael Hahn, Richard Futrell, Roger Levy, and Edward Gibson. 2022. A resource-rational model of human processing of recursive linguistic structure. *Proceedings of the National Academy of Sciences*, 119(43):e2122602119.

John Hale. 2001. A probabilistic Earley parser as a psycholinguistic model. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland. Association for Computational Linguistics.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger Levy. 2020. A systematic assessment of syntactic generalization in neural language models. In *Proceedings of the Association of Computational Linguistics*.

Julie Kallini, Isabel Papadimitriou, Richard Futrell, Kyle Mahowald, and Christopher Potts. 2024. Mission: Impossible language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14691–14714, Bangkok, Thailand. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *The Third International Conference for Learning Representations*, San Diego, California, USA.

Roger Levy. 2008. Expectation-based syntactic comprehension. *Cognition*, 106(3).

Luca Malagutti, Andrius Buinovskij, Anej Svete, Clara Meister, Afra Amini, and Ryan Cotterell. 2024. The role of n-gram smoothing in the age of neural networks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6882–6899, Mexico City, Mexico. Association for Computational Linguistics.

R. Thomas McCoy, Robert Frank, and Tal Linzen. 2020. Does syntax need to grow on trees? sources of hierarchical inductive bias in sequence-to-sequence networks. *Transactions of the Association for Computational Linguistics*, 8:125–140.

Tom M. Mitchell. 1980. The need for biases in learning generalizations. Technical Report CBM-TR 5-110, Rutgers University, New Brunswick, New Jersey, USA.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Steven Piantadosi. 2024. Modern language models refute Chomsky's approach to language. LingBuzz Published In: Edward Gibson & Moshe Poliak (eds.), From fieldwork to linguistic theory: A tribute to Dan Everett (Empirically Oriented Theoretical Morphology and Syntax 15), 353–414. Berlin: Language Science Press. https://doi.org/10.5281/zenodo.12665933.

Jonathan Rawski and Jeffrey Heinz. 2019. No free lunch in linguistics or machine learning: Response to pater. *Language*, 95:e125 – e135.

C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

William Timkey and Tal Linzen. 2023. A language model with limited memory capacity captures interference in human sentence processing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8705–8720, Singapore. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia. Association for Computational Linguistics.

Aditya Yedetore, Tal Linzen, Robert Frank, and R. Thomas McCoy. 2023. How poor is the stimulus? Evaluating hierarchical generalization in neural networks trained on child-directed speech. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9370–9393, Toronto, Canada. Association for Computational Linguistics.

A Probabilistic Finite-state Automata

Before listing a number of useful results for computing quantities of interest in PFSAs, we list a few relevant definitions.

Definition A.1. *Let $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ be a PFSA. We define the **transition matrix** $\boldsymbol{M} \in \mathbb{R}^{|Q| \times |Q|}$ of $\mathcal{A}$ as the matrix containing the probabilities of transitioning from state $q_i \in Q$ to state $q_j \in Q$ in $\mathcal{A}$ with any $y \in \Sigma$:*

$$M_{i,j} \stackrel{\text{def}}{=} \sum_{y \in \Sigma} \sum_{q_i \xrightarrow{y/w} q_j \in \delta} w, \tag{22}$$

*where we fix some arbitrary enumeration of states $(q_1, \ldots, q_{|Q|})$. We also define the **symbol-specific transition matrix** $\boldsymbol{M}^{(y)}$ where $M^{(y)}_{i,j}$ is the probability of transitioning from state $q_i \in Q$ to state $q_j \in Q$ in $\mathcal{A}$ with a y-labeled transition:*

$$M^{(y)}{}_{i,j} \stackrel{\text{def}}{=} \sum_{q_i \xrightarrow{y/w} q_j \in \delta} w. \tag{23}$$

We naturally extend this definition to strings and define for $\boldsymbol{y} = y_1 \cdots y_T$:

$$\boldsymbol{M}^{(\boldsymbol{y})} \stackrel{\text{def}}{=} \boldsymbol{M}^{(y_1)} \cdots \boldsymbol{M}^{(y_T)}. \tag{24}$$

Remark 1. *It is a standard exercise to show that $M^{(\boldsymbol{y})}{}_{i,j}$ equals the sum of the weights of $\boldsymbol{y}$-scanning strings from q_i to q_j.*

Definition A.2. *Let $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ be a PFSA. The **emission matrix** $\boldsymbol{E} \in \mathbb{R}^{|Q| \times |\Sigma|}$ is defined by*

$$E_{i,k} \stackrel{\text{def}}{=} \sum_{q_i \xrightarrow{y_k/w} q' \in \delta} w. \tag{25}$$

For a PFSA $\mathcal{A}$ and a path $\boldsymbol{\pi} = q_0 \xrightarrow{y_1/w_1} \cdots \xrightarrow{y_N/w_N} q_N \in \Pi(\mathcal{A})$, we write $\iota(\boldsymbol{\pi}) \stackrel{\text{def}}{=} q_0$ for the initial state of the path and $\varphi(\boldsymbol{\pi}) \stackrel{\text{def}}{=} q_N$ for its final state. We define the path prefix random variable $\overrightarrow{\Pi}$ distributed as

$$p\left(\overrightarrow{\Pi} = \boldsymbol{\pi}\right) \propto \lambda(\iota(\boldsymbol{\pi}))\overline{\boldsymbol{w}}(\boldsymbol{\pi}). \tag{26}$$

This is analogous to prefix string probabilities and the distribution is normalizable exactly when prefix probabilities are. Similarly, we define $\overleftrightarrow{\Pi}$, which is distributed as

$$p\left(\overleftrightarrow{\Pi} = \boldsymbol{\pi}\right) \propto \sum_{\substack{\boldsymbol{\pi}' \in \Pi(\mathcal{A}) \\ \varphi(\boldsymbol{\pi}') = \iota(\boldsymbol{\pi})}} \lambda\big(\iota\big(\boldsymbol{\pi}'\big)\big)\overline{\boldsymbol{w}}(\boldsymbol{\pi}')\overline{\boldsymbol{w}}(\boldsymbol{\pi}), \tag{27}$$

which is analogous to string infix probabilities.

PFSAs are particularly attractive to study since they allow us to exactly compute many interesting quantities efficiently. In the following section, we describe how one can compute the m-local entropy of the language model defined by a PFSA.

A.1 Useful Properties of Probabilistic Finite-State Automata

The following lemmata hold for a general PFSA $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ and the language model $p_\mathcal{A}$ induced by it. None of the results are novel, but we include full proofs for completeness.

Lemma A.1 (Computing the probability of a string with a PFSA)**.** *The probability of $\boldsymbol{y} \in \Sigma^*$ is:*

$$p_\mathcal{A}(\boldsymbol{y}) = \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{\rho}. \tag{28}$$

Proof. We know from Remark 1 that $\boldsymbol{M}^{(\boldsymbol{y})}_{i,j}$ corresponds to the sum of the path weights from q_i to q_j. Multiplying each entry with the source state's initial weight and the target state's final weight, we arrive at the result. □

Lemma A.2 (Computing the prefix probability of a string)**.** *The prefix probability of $\boldsymbol{y} \in \Sigma^*$ is:*

$$\overrightarrow{p}_\mathcal{A}(\boldsymbol{y}) = \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho}. \tag{29}$$

Proof.

$$\begin{aligned}
\overrightarrow{p}_\mathcal{A}(\boldsymbol{y}) &= \sum_{\boldsymbol{y}' \in \Sigma^*} p_\mathcal{A}(\boldsymbol{y}\boldsymbol{y}') && \text{(30a)}\\
&= \sum_{\boldsymbol{y}' \in \Sigma^*} \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y}\boldsymbol{y}')} \boldsymbol{\rho} && \text{(Lemma A.1, 30b)}\\
&= \sum_{\boldsymbol{y}' \in \Sigma^*} \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^{(\boldsymbol{y}')} \boldsymbol{\rho} && \text{(30c)}\\
&= \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \left(\sum_{\boldsymbol{y}' \in \Sigma^*} \boldsymbol{M}^{(\boldsymbol{y}')} \right) \boldsymbol{\rho} && \text{(30d)}\\
&= \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho} && \text{(30e)}
\end{aligned}$$

□

Lemma A.3 (Computing the next-symbol distribution)**.** *Let $\boldsymbol{y} \in \Sigma^*$. The distribution over the next symbols after observing $\boldsymbol{y}$ is*

$$p_\mathcal{A}\Big(y_k \mid \boldsymbol{s}\Big(\overrightarrow{\Pi}\Big) = \boldsymbol{y}\Big) = \frac{\big(\boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{E}\big)_k}{\overrightarrow{p}_\mathcal{A}(\boldsymbol{y})}. \tag{31}$$

Proof.

$$p_{\mathcal{A}}\Big(y_k \mid \mathbf{s}\Big(\overrightarrow{\Pi}\Big) = \boldsymbol{y}\Big) = \frac{p\left(y_k, \mathbf{s}\Big(\overrightarrow{\Pi}\Big) = \boldsymbol{y}\right)}{p\left(\mathbf{s}\Big(\overrightarrow{\Pi}\Big) = \boldsymbol{y}\right)} \tag{32a}$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y})} \lambda(\iota(\boldsymbol{\pi}))\ \overline{\boldsymbol{w}}(\boldsymbol{\pi})\ p\left(y_k \mid \varphi(\boldsymbol{\pi})\right)$$

(Summing over all $\boldsymbol{y}$-yielding paths, 32b)

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{\substack{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y}) \\ \varphi(\boldsymbol{\pi}) = q_j}} \lambda(\iota(\boldsymbol{\pi}))\ \overline{\boldsymbol{w}}(\boldsymbol{\pi}) \tag{32c}$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \lambda(q_i) \sum_{\substack{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y}) \\ \iota(\boldsymbol{\pi}) = q_i, \varphi(\boldsymbol{\pi}) = q_j}} \overline{\boldsymbol{w}}(\boldsymbol{\pi}) \tag{32d}$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \lambda(q_i) \boldsymbol{M}^{(\boldsymbol{y})}_{i,j} \tag{32e}$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \left(\boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})}\right)_j \tag{32f}$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \sum_{j=1}^{|Q|} \left(\boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})}\right)_j w \qquad (q_i \xrightarrow{y_k/w} q' \in \delta, \text{ 32g})$$

$$= \frac{1}{\overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y})} \left(\boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{E}\right)_k \qquad \text{(Eq. (25), 32h)}$$

□

Lemma A.4 (Computing the infix probability of a string)**.** *The infix probability of $\boldsymbol{y} \in \Sigma^*$ is:*

$$\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{y}) = \boldsymbol{\lambda}^\top \boldsymbol{M}^* \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho}. \tag{33}$$

Proof.

$$\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{y}) = \sum_{\boldsymbol{y}' \in \Sigma^*} \overrightarrow{p}_{\mathcal{A}}(\boldsymbol{y}'\boldsymbol{y}) \tag{34a}$$

$$= \sum_{\boldsymbol{y}' \in \Sigma^*} \boldsymbol{\lambda}^\top \boldsymbol{M}^{(\boldsymbol{y}'\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho} \qquad \text{(Lemma A.2, 34b)}$$

$$= \boldsymbol{\lambda}^\top \left(\sum_{\boldsymbol{y}' \in \Sigma^*} \boldsymbol{M}^{(\boldsymbol{y}')}\right) \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho} \tag{34c}$$

$$= \boldsymbol{\lambda}^\top \boldsymbol{M}^* \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{M}^* \boldsymbol{\rho} \tag{34d}$$

□

Lemma A.5 (Computing the infix next-symbol distribution)**.** *Let $\boldsymbol{c} \in \Sigma^{m-1}$. The distribution over the next symbols after observing $\boldsymbol{c}$ as the last $m-1$ symbols is*

$$p_{\mathcal{A}}\Big(y_k \mid \boldsymbol{s}\Big(\overleftrightarrow{\Pi}\Big) = \boldsymbol{c}\Big) = \frac{\left(\boldsymbol{\lambda}^\top \boldsymbol{M}^* \boldsymbol{M}^{(\boldsymbol{c})} \boldsymbol{E}\right)_k}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})}. \tag{35}$$

Proof.

$$p_{\mathcal{A}}\Big(y_k \mid \mathbf{s}\Big(\overleftrightarrow{\Pi}\Big) = \boldsymbol{c}\Big) = \frac{p\left(y_k, \mathbf{s}\Big(\overleftrightarrow{\Pi}\Big) = \boldsymbol{c}\right)}{p\left(\mathbf{s}\Big(\overleftrightarrow{\Pi}\Big) = \boldsymbol{c}\right)} \tag{36a}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{\boldsymbol{y}' \in \Sigma^*} \sum_{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y}'\boldsymbol{c})} \lambda(\iota(\boldsymbol{\pi}))\, \overline{\boldsymbol{w}}(\boldsymbol{\pi})\, p\left(y_k \mid \varphi(\boldsymbol{\pi})\right)$$

(Summing over all $\boldsymbol{c}$-ending paths, 36b)

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{\boldsymbol{y}' \in \Sigma^*} \sum_{\substack{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y}'\boldsymbol{c}) \\ \varphi(\boldsymbol{\pi}) = q_j}} \lambda(\iota(\boldsymbol{\pi}))\, \overline{\boldsymbol{w}}(\boldsymbol{\pi}) \tag{36c}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \sum_{\boldsymbol{y}' \in \Sigma^*} \lambda(q_i) \sum_{\substack{\boldsymbol{\pi} \in \Pi(\mathcal{A}, \boldsymbol{y}'\boldsymbol{c}) \\ \iota(\boldsymbol{\pi}) = q_i, \varphi(\boldsymbol{\pi}) = q_j}} \overline{\boldsymbol{w}}(\boldsymbol{\pi}) \tag{36d}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \sum_{\boldsymbol{y}' \in \Sigma^*} \lambda(q_i) \left(\boldsymbol{M}^{(\boldsymbol{y}')} \boldsymbol{M}^{(\boldsymbol{c})}\right)_{i,j} \tag{36e}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \lambda(q_i) \sum_{\boldsymbol{y}' \in \Sigma^*} \left(\boldsymbol{M}^{(\boldsymbol{y}')} \boldsymbol{M}^{(\boldsymbol{c})}\right)_{i,j} \tag{36f}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \sum_{i=1}^{|Q|} \lambda(q_i) \left(\boldsymbol{M}^{*} \boldsymbol{M}^{(\boldsymbol{c})}\right)_{i,j} \tag{36g}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} p\left(y_k \mid q_j\right) \left(\boldsymbol{\lambda}^{\top} \boldsymbol{M}^{*} \boldsymbol{M}^{(\boldsymbol{y})}\right)_{j} \tag{36h}$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \sum_{j=1}^{|Q|} \left(\boldsymbol{\lambda}^{\top} \boldsymbol{M}^{*} \boldsymbol{M}^{(\boldsymbol{y})}\right)_{j} w \qquad (q_i \xrightarrow{y_k/w} q' \in \delta, \text{ 36i})$$

$$= \frac{1}{\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})} \left(\boldsymbol{\lambda}^{\top} \boldsymbol{M}^{*} \boldsymbol{M}^{(\boldsymbol{y})} \boldsymbol{E}\right)_{k} \qquad (\text{Eq. (25)}, \text{ 36j})$$

□

Lemma A.6 (Computing the m-local entropy of a DPFSA)**.** *The m-local entropy of $p_{\mathcal{A}}$ can be computed in time $\mathcal{O}\Big(m|Q|^3|\Sigma|^{m-1}\Big)$.*

Proof. M-local entropy of $p_{\mathcal{A}}$ can be computed as

$$\mathrm{H}_m\left(p_{\mathcal{A}}\right) = \mathop{\mathbb{E}}_{\boldsymbol{c} \sim p(\boldsymbol{C} = \boldsymbol{c})} \left[\mathrm{H}(p_{\mathcal{A}}(Y_{\boldsymbol{c}}))\right] \tag{37a}$$

$$= \frac{1}{\overleftrightarrow{Z}} \sum_{\boldsymbol{c} \in \Sigma^{m-1}} \overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c}) \mathrm{H}(p_{\mathcal{A}}(Y_{\boldsymbol{c}})) \tag{37b}$$

The terms $\overleftrightarrow{p}_{\mathcal{A}}(\boldsymbol{c})$ and $p_{\mathcal{A}}(Y_{\boldsymbol{c}})$ can be computed in time $\mathcal{O}\Big(m|Q|^3\Big)$ as per Lemmata A.4 and A.5 for each $\boldsymbol{c} \in \Sigma^{m-1}$. Computing this for each $\boldsymbol{c}$ individually, we arrive at the claimed complexity. □

B Generating Random PFSAs

Algorithm 1 and the subprocedure in Algorithm 2 describe our PFSA generation procedure.

Algorithm 1 Generate a Random DPFSA.

Input: $|Q|$ (number of states), $|\Sigma|$ (number of symbols), μ (target mean string length), R_T (topology random generator), and R_W (weight random generator).

Output: A PFSA A with randomly assigned transitions and normalized weights with $|Q|$ states and $|\Sigma|$ symbols.

Note: $\textsc{Choice}(R, S, 1)$ denotes selecting one element uniformly at random from the set S using the random generator R.
`unused` is initialized as Q, and `out-arcs` is a mapping that assigns to each state a subset of Σ (the allowed outgoing symbols).

For exponential sampling, we write $w \sim \mathrm{Exp}(0.1)$ to denote that w is drawn from an exponential distribution with rate 0.1, i.e., with density $f(w) = 0.1\, e^{-0.1w}$ for $w \geq 0$.

function RANDOMDPFSA($|Q|,\ |\Sigma|,\ \mu,\ R_T,\ R_W$)
  $q_\iota \leftarrow \textsc{Choice}(R_T, Q, 1)$
  Initialize $\mathcal{A} \leftarrow (\Sigma, Q, \delta, \lambda, \rho)$
  Initialize $\boldsymbol{\lambda} \leftarrow \mathbf{0}_{|Q|}$ and **set** $\lambda(q_\iota) \leftarrow 1$
  Initialize $\boldsymbol{M}^{(y)}$ to a $|Q| \times |Q|$ matrix of zeros for $y \in \Sigma$
  `unused` $\leftarrow Q$
  `state-outgoing-symbols` $\leftarrow \textsc{GetOutgoingSymbols}(Q,\ \Sigma,\ R_T)$
  for $q \in Q$ **do**
    for $y \in \Sigma$ **do**
      if `unused` $\neq \emptyset$ **then**
        $q' \leftarrow \textsc{Choice}(R_T, \texttt{unused}, 1)$
        Remove q' from `unused`
      else
        $q' \leftarrow \textsc{Choice}(R_T, Q, 1)$
      Let $w \sim \mathrm{Exp}(0.1)$
      ${M^{(y)}}_{q,q'} \leftarrow w \cdot \mathbb{1}\{y \in \texttt{state-outgoing-symbols}[q]\} + 0.001$
  for $q \in Q$ **do** ▷ Set final weights and normalize outgoing weights for each state.
    $t \leftarrow \sum_{y=0}^{|\Sigma|-1} \mathrm{sum}({\boldsymbol{M}^{(y)}}_{q,:})$
    $\rho(q) \leftarrow t/\mu$
    $s \leftarrow t + \rho(q)$
    for $y \in \{0, \ldots, |\Sigma| - 1\}$ **do**
      ${\boldsymbol{M}^{(y)}}_{q,:} \leftarrow {\boldsymbol{M}^{(y)}}_{q,:}/s$
    $\rho(q) \leftarrow \rho(q)/s$
  return $\mathcal{A}$

Algorithm 2 Generate Outgoing Symbols for Each State.

Input: $|Q|$ (number of states), $|\Sigma|$ (number of symbols), R (random generator), and s_{min} (min. unique symbols per state; default: 2).

Output: A list S of $|Q|$ sets, each containing outgoing symbols.

Note: $\textsc{Choice}(R, X, k)$ selects k distinct elements uniformly at random from X using R, and $\textsc{Integers}(R, a, b)$ returns a random integer in $[a, b)$.

function $\textsc{GetOutgoingSymbols}(Q, \Sigma, R, s_{min})$
  Initialize `state-outgoing-symbols` $\leftarrow$ an array of $|Q|$ empty sets
  for $q \in Q$ **do**
    `s` $\leftarrow \textsc{Choice}(R, \mathcal{N}, \min(s_{min}, |\Sigma|))$ ▷ Assign each state at least s_{min} symbols.
    `state-outgoing-symbols`$[q] \leftarrow$ `state-outgoing-symbols`$[q] \cup$ `s`
  for $y \in \Sigma$ **do** ▷ Ensure each symbol appears in at least one set.
    $q \leftarrow \textsc{Choice}(R, Q, 1)$
    Add y to `state-outgoing-symbols`[q]
  `M` $\leftarrow \sum_{q=0}^{|Q|-1} \textsc{Integers}(R, 0, \max(1, \lfloor |\Sigma|/2 \rfloor - s_{min}))$
  for $j \leftarrow 1$ **to** `M` **do** ▷ Add random transitions.
    $y \leftarrow \textsc{Choice}(R, \Sigma, 1)$
    $q \leftarrow \textsc{Choice}(R, Q, 1)$
    Add y to `state-outgoing-symbols`[q]
  return `state-outgoing-symbols`

C Details of Neural Language Models

C.1 Transformer

We use a 4-layer causally-masked transformer with 768-dimensional embeddings, 3,072-dimensional feedforward layers, and 12 attention heads, implemented in PyTorch. Following Vaswani et al. (2017), we map input symbols to vectors of size 768 with a scaled embedding layer and add sinusoidal positional encodings. We use pre-norm instead of post-norm and apply layer norm to the output of the last layer. We use the same dropout rate throughout the transformer. We apply it in the same places as Vaswani et al. (2017), and, as implemented by PyTorch, we also apply it to the hidden units of feedforward sublayers and to the attention probabilities of scaled dot-product attention operations. We always use BOS as the first input symbol to the transformer.

C.2 LSTM

We use a single-layer LSTM (Gers and Schmidhuber, 2001) with 512-dimensional hidden units, implemented in PyTorch with some modifications as in Butoi et al. (2025). Figure 4 shows our definition of the LSTM architecture. Here, $\boldsymbol{E}$ is an embedding matrix to map each symbol w_t of the input string to an embedding $\boldsymbol{x}_t = \boldsymbol{E}_{w_t}$. The size of the embeddings is always d, the size of the hidden vectors, and we denote the number of layers in the model as L. Also, $\odot$ denotes elementwise multiplication, and $\textsc{Dropout}(\cdot)$ indicates the application of dropout. Here, $\boldsymbol{w}_0^{(\ell)} \in \mathbb{R}^d$ is a learned parameter, making the initial hidden state $\boldsymbol{h}_0^{(\ell)}$ of each layer learned. A modification is made from the original PyTorch implementation: each pair of b_{ii} and b_{hi}, b_{if} and b_{hf}, b_{ig} and b_{hg}, and b_{io} and b_{ho} is replaced with a single bias parameter per layer.

D Hyperparameters for Neural Language Model Training

Wherever dropout is applicable, we use a dropout rate of 0.1. For layer norm, we initialize weights to 1 and biases to 0. We initialize all other parameters by sampling uniformly from $[-0.1, 0.1]$.

For each epoch, we randomly shuffle the training set and group strings of similar lengths into the same minibatch, enforcing an upper limit of 2,048 symbols per batch, including padding, BOS, and EOS symbols. We train each model by minimizing cross-entropy on validation set using Adam (Kingma and Ba, 2015). We clip gradients with a threshold of 5 using L^2 norm rescaling. We take a checkpoint every 10k examples, at which point we evaluate the model on the validation set and update the learning rate and early stopping schedules. We multiply the learning rate by 0.5 after 5 checkpoints of no decrease in cross-entropy on the validation set, and we stop early after 10 checkpoints of no decrease. We select the checkpoint with the lowest cross-entropy on the validation set when reporting results. We train for a maximum of 1k epochs.

E Additional Experimental Results for Experiment 1

E.1 How Does *M*-local Entropy Affect LM Performance?

Table 2 reports the Pearson correlation coefficients between the m-local entropy of PFSA and the estimated KL divergence ($\hat{D}_{\mathrm{KL}}$; §4.1.2).

E.2 Experiments with BabyLM Corpus

We also conducted the same set of experiments using the BabyLM corpus (Choshen et al., 2024). Table 3 and Figure 5 show the experimental results. They show the same trends as in our main experiment (§3), but with slightly different tendencies for the REVERSE language.

F Computational Resources

Across all experiments, we used a total of approximately 717.5 GPU hours. Training was conducted on NVIDIA GeForce RTX 4090 24GB and NVIDIA Quadro RTX 6000 24GB GPUs.

G License of the Data

The BLLIP corpus (Charniak et al., 2000) is used under the terms of the BLLIP 1987-89 WSJ Corpus Release 1 License Agreement.

$$\boldsymbol{h}_t^{(0)} \stackrel{\text{def}}{=} \boldsymbol{x}_t = \boldsymbol{E}_{w_t} \qquad (1 \le t \le n) \tag{38a}$$

$$\boldsymbol{h}_0^{(\ell)} \stackrel{\text{def}}{=} \tanh(\boldsymbol{w}_0^{(\ell)}) \qquad (1 \le \ell \le L) \tag{38b}$$

$$\not{\boldsymbol{h}}_t^{(\ell)} \stackrel{\text{def}}{=} \textsc{Dropout}(\boldsymbol{h}_t^{(\ell)}) \qquad (0 \le \ell \le L; 0 \le t \le n) \tag{38c}$$

$$\boldsymbol{i}_t^{(\ell)} \stackrel{\text{def}}{=} \sigma(\boldsymbol{W}_{\text{i}}^{(\ell)} \begin{bmatrix} \not{\boldsymbol{h}}_t^{(\ell-1)} \\ \boldsymbol{h}_{t-1}^{(\ell)} \end{bmatrix} + \boldsymbol{b}_{\text{i}}^{(\ell)}) \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38d}$$

$$\boldsymbol{f}_t^{(\ell)} \stackrel{\text{def}}{=} \sigma(\boldsymbol{W}_{\text{f}}^{(\ell)} \begin{bmatrix} \not{\boldsymbol{h}}_t^{(\ell-1)} \\ \boldsymbol{h}_{t-1}^{(\ell)} \end{bmatrix} + \boldsymbol{b}_{\text{f}}^{(\ell)}) \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38e}$$

$$\boldsymbol{g}_t^{(\ell)} \stackrel{\text{def}}{=} \tanh(\boldsymbol{W}_{\text{g}}^{(\ell)} \begin{bmatrix} \not{\boldsymbol{h}}_t^{(\ell-1)} \\ \boldsymbol{h}_{t-1}^{(\ell)} \end{bmatrix} + \boldsymbol{b}_{\text{g}}^{(\ell)}) \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38f}$$

$$\boldsymbol{o}_t^{(\ell)} \stackrel{\text{def}}{=} \sigma(\boldsymbol{W}_{\text{o}}^{(\ell)} \begin{bmatrix} \not{\boldsymbol{h}}_t^{(\ell-1)} \\ \boldsymbol{h}_{t-1}^{(\ell)} \end{bmatrix} + \boldsymbol{b}_{\text{o}}^{(\ell)}) \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38g}$$

$$\boldsymbol{c}_t^{(\ell)} \stackrel{\text{def}}{=} \boldsymbol{f}_t^{(\ell)} \odot \boldsymbol{c}_{t-1}^{(\ell)} + \boldsymbol{i}_t^{(\ell)} \odot \boldsymbol{g}_t^{(\ell)} \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38h}$$

$$\boldsymbol{h}_t^{(\ell)} \stackrel{\text{def}}{=} \boldsymbol{o}_t^{(\ell)} \odot \tanh(\boldsymbol{c}_t^{(\ell)}) \qquad (1 \le \ell \le L; 1 \le t \le n) \tag{38i}$$

$$\boldsymbol{c}_0^{(\ell)} \stackrel{\text{def}}{=} \boldsymbol{0} \qquad (1 \le \ell \le L) \tag{38j}$$

$$\boldsymbol{h}_t \stackrel{\text{def}}{=} \not{\boldsymbol{h}}_t^{(L)} \qquad (0 \le t \le n) \tag{38k}$$

Figure 4: Definition of the LSTM architecture employed in this work, following the formulation in **?**.

	$\lvert Q\rvert$	16			24			32		
	$\lvert\Sigma\rvert$	32	48	64	32	48	64	32	48	64
ARCHITECTURE	M									
LSTM	**2**	0.433	0.501	0.137	0.291	0.583	0.121	0.423	0.311	0.623
	3	0.396	0.532	0.230	0.291	0.488	0.119	0.460	0.274	0.702
	4	0.412	0.546	0.236	0.311	0.477	0.122	0.474	0.283	0.702
	5	0.415	0.554	0.234	0.338	0.472	0.120	0.466	0.283	0.686
TRANSFORMER	**2**	0.740	0.679	0.455	0.290	0.658	0.844	0.551	0.622	0.709
	3	0.743	0.728	0.569	0.374	0.735	0.859	0.693	0.832	0.820
	4	0.737	0.674	0.549	0.389	0.727	0.844	0.668	0.848	0.799
	5	0.717	0.614	0.516	0.333	0.705	0.830	0.625	0.833	0.770

Table 2: Pearson correlation coefficients between m-local entropy and KL divergence for different architectures, number of states $|Q|$, and alphabet sizes $|\Sigma|$.

	2-local entropy	3-local entropy	4-local entropy	5-local entropy
BASE	5.78	3.72	2.69	2.32
REVERSE	6.50	3.87	2.78	2.43
EVENODDSHUFFLE	6.82	4.47	3.36	3.14
ODDEVENSHUFFLE	6.94	4.45	3.39	3.14
LOCALSHUFFLE (K=3)	6.99 ± 0.16	4.27 ± 0.08	3.21 ± 0.07	2.97 ± 0.08
LOCALSHUFFLE (K=4)	7.09 ± 0.15	4.35 ± 0.06	3.25 ± 0.03	3.06 ± 0.04
LOCALSHUFFLE (K=5)	7.15 ± 0.13	4.42 ± 0.06	3.29 ± 0.04	3.08 ± 0.03
LOCALSHUFFLE (K=6)	7.25 ± 0.11	4.47 ± 0.07	3.35 ± 0.06	3.14 ± 0.05
LOCALSHUFFLE (K=7)	7.28 ± 0.12	4.50 ± 0.08	3.39 ± 0.07	3.19 ± 0.08
DETERMINISTICSHUFFLE	7.41	4.69	3.59	3.40

Table 3: M-local entropy values for BASE (original) corpus and different transformed corpora. "Local shuffle" refers to the K-LOCALDETERMINISTICSHUFFLE. Values are shown as mean ± standard deviation (averaged over different random seeds).

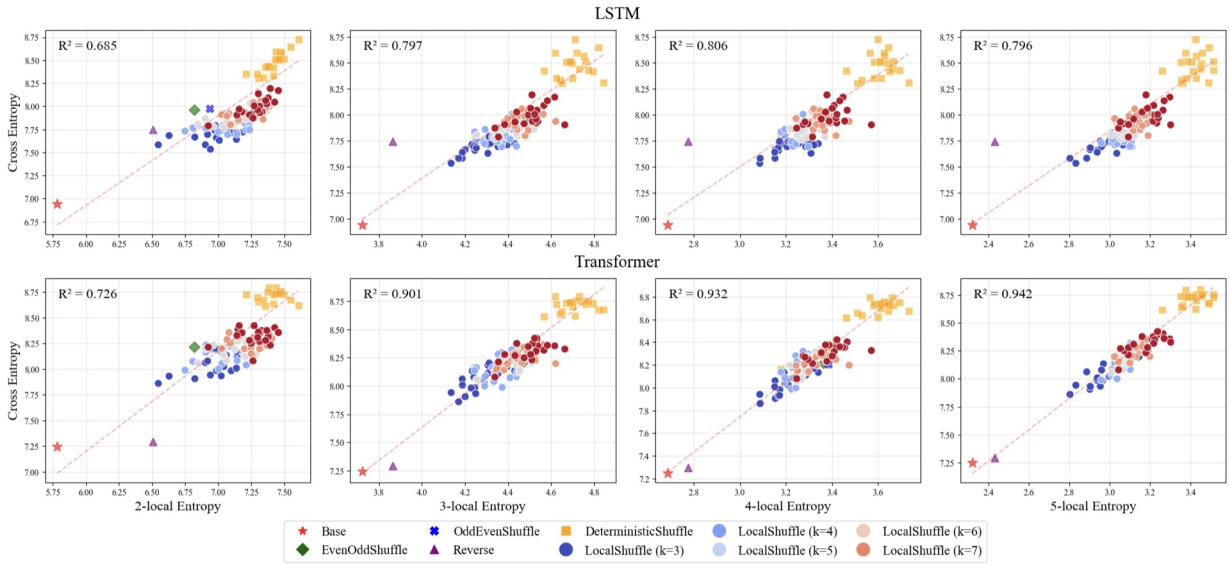

Figure 5: Scatter plots of next-symbol cross-entropy (y-axis) versus m-local entropy (x-axis) for $m \in \{2, 3, 4, 5\}$, for both LSTM (top row) and causally-masked Transformer encoder (Transformer; bottom row) models. Each marker type/color corresponds to a different perturbation (e.g., `Reverse`, `DeterministicShuffle`, K-LOCALDETERMINISTICSHUFFLE with various window sizes, etc.). The red star indicates the unperturbed BASE condition (original corpus). The dashed line in each panel is a linear fit, with R^2 indicating the coefficient of determination.