Lexinvariant Language Models

Qian Huang[1]
qhwang@cs.stanford.edu

Eric Zelikman[1]
ezelikman@cs.stanford.edu

Sarah Li Chen[1]
sachen@stanford.edu

Yuhuai Wu[12]
yuhuai@cs.stanford.edu

Gregory Valiant[1]
gvaliant@cs.stanford.edu

Percy Liang[1]
pliang@cs.stanford.edu

[1]Stanford University
[2]Google Research

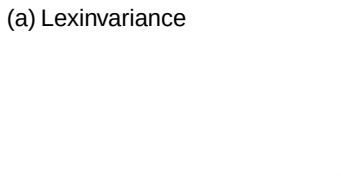
Abstract

Token embeddings, a mapping from discrete lexical symbols to continuous vectors, are at the heart of any language model (LM). However, lexical symbol meanings can also be determined and even redefined by their structural role in a long context. In this paper, we ask: is it possible for a language model to be performant without *any* fixed token embeddings? Such a language model would have to rely entirely on the co-occurence and repetition of tokens in the context rather than the *a priori* identity of any token. To answer this, we study *lexinvariant* language models that are invariant to lexical symbols and therefore do not need fixed token embeddings in practice. First, we prove that we can construct a lexinvariant LM to converge to the true language model at a uniform rate that is polynomial in terms of the context length, with a constant factor that is sublinear in the vocabulary size. Second, to build a lexinvariant LM, we simply encode tokens using random Gaussian vectors, such that each token maps to the same representation within each sequence but different representations across sequences. Empirically, we demonstrate that it can indeed attain perplexity comparable to that of a standard language model, given a sufficiently long context. We further explore two properties of the lexinvariant language models: First, given text generated from a substitution cipher of English, it implicitly implements Bayesian in-context deciphering and infers the mapping to the underlying real tokens with high accuracy. Second, it has on average 4X better accuracy over synthetic in-context reasoning tasks. Finally, we discuss regularizing standard language models towards lexinvariance and potential practical applications.

1 Introduction

All language processing systems rely on a stable lexicon, which assumes that a token (a word or subword such as *tree*) has a consistent contribution to the meaning of a text (though of course this meaning is mediated by context). In neural language models (LMs), this contribution is the token embedding, which *stably* maps each token into a continuous vector [21, 16, 17, 7, 6]. However, in real language, a token's contribution might be determined by its structural role; in math and code, novel variable names are arbitrarily defined to carry new meaning, and poems such as Jabberwocky exploit humans' lexical flexibility in interpreting novel words such as *vorpal*. Besides standard language understanding, this lexical flexibility also correlates with a stronger in-context reasoning performance. For example, GPT-3 [6] and other large language models that demonstrate high lexical flexibility show strong performance on tasks involving in-context reasoning over new concepts and rules.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).



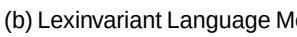


Figure 1: De nition (a) and construction (b) of lexinvariant language model

Motivated by the above, we ask whether we can push this exibility to the extreme: can we build a language model without any stable lexical mapping? To this end, we formulate and study such lexinvariant language models. We de ne a lexinvariant language model as a language model that assigns the same probability to all lexical permutations of a sequence. Formally, we de ne a lexical permutation to be a one-to-one mapping of a set of lexical symbols[1] onto itself. Then the lexinvariant language model is de ned as a language model over the symbol sequences $x_1, \ldots, x_n$ with the following property:

$$p(x_1, \ldots, x_n) = p(\pi(x_1), \ldots, \pi(x_n)) \quad \forall \pi \tag{1}$$

For example, a lexinvariant language model (whose vocabulary is letters and space) should assign the same probability to the phrase "a big banana" as "e cop cekeke" because the two are the same up to the permutation $\pi = \{a:e, b:c, i:o, n:k, g:p, \ldots\}$ (Figure 1a).

The central question is: how well can lexinvariant language models predict the next token given an increasingly long context? We nd the answer is almost as well as standard language models, both theoretically and empirically. This is rather surprising given that lexinvariance seems like a strong limitation (a model doesn't know what any individual symbol means!). However, the intuition is that given longer contexts, a lexinvariant model can both infer the latent permutation (fully) up to whatever ambiguity is present in the language model, and do the standard next word prediction task jointly.

Theoretically, we prove that a constructed lexinvariant language model can converge to the true language model as the context length increases—that is, the average L1 distance between the predictions of the two models decreases with a convergence rate of $\tilde{O}(\frac{d}{T})^{\frac{1}{4}}$, where T is the length of the context and d is the vocabulary size, and where the big-O notation hides polylogarithmic factors of d and T and an absolute constant that is independent of the language model.

Empirically, we train a lexinvariant LM by replacing standard embeddings in a decoder-only Transformer [24] with per-sequence random Gaussian vectors, such that the same symbols get the same embedding within each sequence but get different embedding across sequences (Figure 1b). We indeed see that the perplexity gap between the lexinvariant LM and the standard LM shrinks as context length increases, as shown in Section 3.2. With a 150M parameters Transformer and a small character-level vocabulary (130 tokens), the average perplexity gap shrinks from 9X to less than 1X the average perplexity of a standard LM after observing 512 tokens over The Pile. With a larger 32K vocabulary, the gap also shrinks, especially on the more structured text like GitHub code, albeit at a much slower rate.

We then explore two additional properties of the lexinvariant LM: in-context deciphering and symbol manipulation. First, we show that given a ciphertext generated by applying a substitution cipher to English text, the lexinvariant LM can be seen as implicitly approximating Bayesian inference of the lexical permutation, i.e., cipher key, in-context. To show this empirically, we train a small MLP probe on top of a frozen pretrained lexinvariant LM to predict the deciphered token corresponding to the last

[1] We speci cally consider lexical symbols as tokens, not necessarily words or other linguistic units.

seen cipher token. We can then read out the inferred cipher key with each pre x of the sequence. We show that the accuracy of this inferred cipher key quickly improves as context length grows, reaching 99.6% average accuracy. We also show examples in Section 3.4 that visualize the uncertainties over different possible lexical mappings maintained by the lexinvariant LM when the cipher key is ambiguous and that the semantic meaning of a symbol with very rare occurrence can be inferred ef ciently relative to other common symbols in context. Second, we show that lexinvariant models perform better than traditional models over synthetic pure in-context reasoning tasks that involve symbol manipulation. We observe a signi cant 4X improvement over a standard language model.

While the primary motivation of this paper is scienti c exploration of a new idea, lexinvariance, we were also curious to see if it could help improve certain tasks, generalizing the performance gain we see on synthetic tasks. We stress that for most practical applications, lexinvariance is far too strong, so these experiments are intended to be illustrative rather than be a recipe for improving state-of-the-art. We discuss potential approaches to integrate the idea of lexinvariant LM into standard language modeling as a form of regularization, such that the LM assumes some form of partially stable symbol representations. The resulting LM can improve upon a standard language model over some BIG-bench tasks [23].

2 Lexinvariant Language Model

We de ne a language model as a probability distribution $p(x_1;\cdots;x_n)$ over input token sequences $x_1;\cdots;x_n \in V^n$, where V is some vocabulary over symbols. A language model is lexinvariant if for all permutations $\sigma : V \to V$ and for all token sequences $x_1;\cdots;x_n \in V^n$, $p(x_1;\cdots;x_n) = p(\sigma(x_1);\cdots;\sigma(x_n))$. For example, if $V = \{a;b\}$, then the model should assign the same probability to aab and bba. One example p that satis es this could simply be

$$p(x) = \begin{cases} 1/2 & x \in \{aab;bba\} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Can such a lexinvariant language model predict language well, even though it can only make next token predictions based on the structure of co-occurence and repetition of input tokens in a single context?

2.1 Convergence on Language Modeling Performance

We show that we can construct a lexinvariant LM (as shown in Figure 2) to model the true language distribution faithfully, given a long enough context. The lexinvariant language model can essentially infer back the latent permutation as it observes more symbols.

Figure 2: Probabilistic graphical model for the lexinvariant LM associated with the true language distribution $p(x_1;\cdots;x_n)$.

As an intuitive example, suppose that $V = \{a;b\}$ and the true language only contains two sequences $babbbb$ and $ababab$ (and their pre xes) with even probability. When given only the rst three letters, a lexinvariant model can't tell the latent permutation and can only assign the same probability to b for the next letter: Due to the lexinvariant property, it assigns the same probability $p(b|bab) = p(b|aba)$ as well as to $p(b|aba) = p(a|bab)$. Further, $p(a|aba) = p(b|aba)$ because the permutations to pre xes aba and bab are equally probable. In contrast, when considering the pre x $abab$, the fourth letter resolves the ambiguity in possible permutations. (Since $baba$ is not in the true language distribution, σ cannot map a to b.) Therefore, the model can correctly predict that $p(a|abab) = 1$ and $p(b|abab) = 0$.

Formally, for a given language model p, we de ne the associated lexinvariant language model $p^0(x_1;\cdots;x_n)$ as $E_\sigma[p(\sigma^{-1}(x_1);\cdots;\sigma^{-1}(x_n))]$. Analyzing it, we have the following theorem:

Theorem 2.1. Let $x_1;\cdots;x_n$ be any token sequence generated by an arbitrary language distribution p with an alphabet of size d. Let $p^0(x_1;\cdots;x_n) = E_\sigma[p(\sigma^{-1}(x_1);\cdots;\sigma^{-1}(x_n))]$. Then, for any

$0 < \epsilon < 1/2$,

$$\frac{1}{T}\sum_{t=1}^{T} \mathrm{KL}(p(x_t \mid x_1, \ldots, x_{t-1}) \,\|\, p^0(x_t \mid x_1, \ldots, x_{t-1}))$$

with probability greater than $1 - \delta$, when $T \geq \frac{d}{\epsilon^4}\,\mathrm{polylog}(d, \frac{1}{\epsilon}, \frac{1}{\delta})$; where the polylogarithmic term hides an absolute constant that is independent of p.

This theorem says that this associated lexinvariant language model converges to modeling the true language distribution fairly efficiently—with polynomial rate and near-linear dependence on vocabulary size d. Strikingly, this holds irrespective of the properties of the language distribution p[2]. In other words, a language model can indeed infer the operational meaning of the tokens in context based solely on the structure of the symbols.

We give a complete proof of this theorem in Appendix A. At a high level, this convergence happens because at most timesteps t, the new observation x_t either provides new information about the permutation σ, or x_t has similar likelihood under the permutations that are likely given $x_1, \ldots, x_{t-1}$. In the simplest case, if the posterior $P(\sigma \mid x_1, \ldots, x_n)$ concentrates on the correct σ, then we converge to the standard LM. But even if it doesn't, that means the uncertainty about σ should not matter for predicting the next token. We make this precise by interpreting $p^0(x_t \mid x_1, \ldots, x_{t-1})$ as performing a multiplicative weights algorithm with the Hedge strategy of Freund and Schapire [8], and then relate the regret bounds to the average KL divergence between the predictions of p and p^0, and ultimately the average L_1 distance between these predictions.

2.2 In-context Bayesian Deciphering

We can see the associated lexinvariant language model as implicitly learning to approximate an in-context Bayesian deciphering process, i.e. inferring a probability distribution over possible lexical permutations based on seen tokens, with the language modeling prior:

$$\begin{aligned} & p^0(x_{n+1} \mid x_1, \ldots, x_n) \\ &= \sum_{\sigma} \frac{1}{d!} \frac{p(\sigma^{-1}(x_1), \ldots, \sigma^{-1}(x_{n+1}))}{p^0(x_1, \ldots, x_n)} \\ &= \sum_{\sigma} \frac{p(\sigma^{-1}(x_1), \ldots, \sigma^{-1}(x_{n+1}))}{p(\sigma^{-1}(x_1), \ldots, \sigma^{-1}(x_n))} \frac{\frac{1}{d!} p(\sigma^{-1}(x_1), \ldots, \sigma^{-1}(x_n))}{p^0(x_1, \ldots, x_n)} \qquad (3) \\ &= \sum_{\sigma} \underbrace{p(\sigma^{-1}(x_{n+1}) \mid \sigma^{-1}(x_1), \ldots, \sigma^{-1}(x_n))}_{\text{language modeling}} \underbrace{P(\sigma \mid x_1, \ldots, x_n)}_{\text{inferring lexical permutation}} \end{aligned}$$

As shown above, p^0 can be reduced to two parts, where the first part is normal language modeling and the second part is the probability distribution of lexical permutations based on seen tokens. So the lexinvariant language model is implicitly learning to model $P(\sigma \mid x_1, \ldots, x_n)$.

We can make this approximate in-context Bayesian deciphering explicit by training a small probe to predict $P(\sigma \mid x_1, \ldots, x_n)$ given the internal representation of the lexinvariant language model. We will show that this indeed recovers σ reasonably accurately in the experiment section.

2.3 Constructing a Lexinvariant Language Model

We now consider how to construct a lexinvariant LM in practice. A typical neural language model, such as a Transformer, converts input tokens to continuous vectors using token embedding and then passes these vectors as input to the rest of the neural network. Thus, the language model p parameterizes depends on the token embedding $E = V \to \mathbb{R}^d$:

$$p(x_1, \ldots, x_n) = T(E(x_1), \ldots, E(x_n)) \qquad (4)$$

To make a neural LM lexinvariant, we can replace the standard stable token embedding with a randomized E and take the expectation over E. Each token $x \in V$ has an independent embedding

[2] The convergence rate could be better depending on the language distribution, such as on math and code, where the symbols should have clear functional meaning in context. We explore this empirically in the experiment section.

$E(x) \sim N(0; I_d)$, and the language model becomes

$$p(x_1;\dots;x_n) = E[T(E(x_1);\dots;E(x_n))] \quad (5)$$

Since $E \overset{d}{=} EP$, the right-hand side is the same when P are applied with any permutation, i.e., for any $x_1;\dots;x_n$:

$$E[T(E(x_1);\dots;E(x_n))] = E[T(E(\pi(x_1));\dots;E(\pi(x_n)))]; \quad (6)$$

showing that the Transformer with random E is lexinvariant as in Eq. 1. Now we can train this lexinvariant LM similarly to a standard LM. Concretely, we sample a new E for each training sequence and minimize the standard language modeling loss as in a standard neural LM. Here we are stochastically optimizing a variational lower bound of the standard language modeling loss with this randomized model by taking the expectation to the outside of the loss over log likelihood. Effectively, the same token gets the same random embedding within each training sequence, but different embedding across training sequences.

In practice, we focus on training decoder-only Transformers with a next token prediction objective in this work, where the model directly models $p(x_{n+1} \mid x_1;\dots;x_n)$ instead of the joint distribution. Our definitions and analysis above still hold in general. The only modification is that the final readout matrix also needs to be replaced with the same E, so that the Transformer can predict the embedding of the next token based on the embeddding of input tokens.

3 Experiments

3.1 Setup

Architecture. For all experiments, we use decoder-only Transformer architecture with T5 relative position bias [19]. We use models with 150M parameters, with 12 layers, 8 heads, head dimension 128, and MLP dimension 4096.

Training. We use the Adafactor optimizer [21], with a cosine decay learning rate schedule [18] from 0.01 to 0.001 based on preliminary experiments. We train the models from scratch for 250K steps on all the settings, with 512 sequence length and 64 batch size. We ran all of our experiments on 8 TPU cores. Our models are implemented in JAX [5].

Datasets. For datasets, we mainly use the Pile [9], a large open-source corpus that contains text collected from 22 diverse high-quality sources. We also run experiments on two additional datasets to explore their effects on the behavior of lexinvariant models: Wiki-40B [8], which contains high quality processed Wikipedia text in 40+ languages, and GitHub (subset of the Pile), which contains code from GitHub repositories with more than 100 stars and less than 1GB files.

3.2 Convergence to Standard Language Models

We first show empirically that lexinvariant LMs can mostly recover the next token prediction performance of standard LMs after a long enough context. As already discussed in section 2.1, the lexinvariant LM will theoretically converge to a standard LM as the context becomes long enough to resolve ambiguity. Here we verify this experimentally and show the variation of this convergence across corpora and tokenizations.

To show this, we train lexinvariant and standard LMs with both character-level vocabulary (128 ascii characters) and T5 default vocab (32k tokens) over the three datasets. For each model, we measure

Figure 3: Perplexity over the Pile with character-level vocabulary (left) and T5 default vocab (right).

the perplexity of each token in each sequence w.r.t. context length, smoothed by moving average within each sequence, i.e. $PPL(x_i;...;x_{i+k}|x_1;...;x_i)^{\frac{1}{k}}$ for context length i. We set the moving average window $k = 100$. We plot results over 100 sequences. As shown in figure 3, the perplexity gap between lexinvariant LM and standard LM gradually shrinks as the prefix becomes longer and longer, albeit much more slowly with a larger vocabulary. This makes intuitive sense since a larger vocabulary has more possibilities of permutations and requires many more prefix tokens to disambiguate. For the 32K vocabulary, the 512 context length will only allow the model to see a very small number of tokens, let alone to see tokens more than once. Nonetheless, the model still manages to show the trend of convergence, since even a small number of repetitions can form common patterns in grammar (such as the usage of spaces, punctuation, articles, etc). For the character-level vocabulary, the perplexity gap shrinks from 9X to less than 1X the average perplexity of the standard LM. With a context length of 511, the lexinvariant LM converges to perplexity 3.38, almost comparable to the perplexity of the standard LM of 2.00. Additionally, we observe that the gap shrinks significantly faster for models trained over Github than standard English text like Wiki-40B since code is more structured and it is easier to decipher the token permutation. We show the comparison across different datasets in Figure 7 in Appendix.

3.3 Recovering Substitution Ciphers

Here we show that lexinvariant LM is implicitly performing Bayesian in-context deciphering by testing its ability to recover cipher keys (e.g. Figure 4a) from character-level substitution ciphers (e.g. `kvR5W 4mfzd @f| Svcgn fw;m uCRmu;;d }%:fBn`). For the lexinvariant LM, this cipher text is perceived as the same text as `the quick brown fox jumps over thirteen lazy dogs`, due to the lexinvariant property. It will then proceed to complete the cipher text with `%%d: uC; @f|` with the same probability as it will complete the normal text with `and the fox`.

Because of this, we cannot directly read out the distribution of possible cipher keys $P(key|x_1;...;x_{n-1})$ implicitly inferred by the lexinvariant LM. To do this, we train a small two-layer MLP probe on top of a frozen trained lexinvariant LM. For each training sequence, we first embed the input sequence with a randomly sampled token embedding E as described in section 2.3 and obtain the hidden activation of the final layer generated by the frozen lexinvariant LM. Then, we pass this activation through the two-layer MLP probe. Finally, instead of decoding the output activations to classification logits with the same E as in the lexinvariant LM, we instead use another learnable non-randomized token embedding matrix E' so that the probe can recover the deciphered token with stable token embeddings. Overall, we train the probe jointly with this embedding matrix E' to predict the current token. Effectively, we are training the probe to decipher the current token using the representation provided by the lexinvariant LM. We train the probe over the same corpus as the original lexinvariant LM for 10k steps. With this probe, we can directly visualize $P(x_n|x_1;...;x_n)$ inferred by the lexinvariant LM, which is effectively one row in the permutation matrix representing

Now we can use this probe to explicitly recover the cipher key. An example ground truth cipher key that we want to recover is shown in Figure 4a. Note that although the substitution cipher is only among lowercase letters, the character-level lexinvariant model we use assumes that all permutations among the 128 characters are possible, making the deciphering even more challenging.

(a) Ground truth (b) Majority vote prediction (c) Cipher key prediction accuracy

Figure 4: (a) (b): Cipher key matrix, where the vertical axis shows the cipher characters and the horizontal axis shows the deciphered letters. The highlighted entries show the correspondences between cipher characters and the actual letters, e.g. `%` deciphers to `t`. (c): Cipher key prediction accuracy, averaged across 1000 input sequences. Context length denotes the start index of the window.

Figure 5: Predicted cipher key for windows of size 50, at indices 0, 50, 100, 200, and 400. Generated using temperature of $T = 1$.

Concretely, we first input ciphertext through the frozen lexinvariant LM with the probe to produce a deciphered sequence. We then select a window of size 100 in the middle of the sequence and perform a majority vote over the corresponding deciphered tokens of each cipher token seen in this window. This essentially produces a predicted cipher key matrix for each window, and we can measure its precision against the ground truth. As shown in Figure 4c, such a cipher key prediction generally has increasingly higher precision as the window is selected later in the context, and it becomes near-perfect by the end of the sequence. Specifically, the cipher key matrix produced by the last window has an average precision of 99.6% over 1000 input sequences.

Finally, we aggregate over the last window of the 1000 sequences to recover a full cipher key, in case certain letters never appear in the last window of certain sequences. We again recover a full cipher key via majority vote. In Figure 4b, we show the highly accurate predicted cipher key recovered from ciphertext produced using the example ground truth cipher key in Figure 4a.

To perform a more detailed analysis showing the Bayesian deciphering process of the lexinvariant model, we use the logits of the probe to recover the predicted distribution of the cipher key $P(\cdot \mid x_1, \ldots, x_{n-1})$. Instead of taking the majority vote of the predicted decipher tokens in the window, we take the mean of logits predicted for each ciphered token. This essentially gives a locally averaged predicted distribution of cipher key matrices. Specifically, the cipher key matrices are generated across windows of 50 characters, and the probabilities are averaged over 1000 input sequences encoded using the same ground truth cipher. As shown in Figure 5, the predicted distribution of cipher key matrix becomes sharper as the prefix becomes longer.

3.4 In-context Bayesian Deciphering Examples

Here, we show several qualitative examples of in-context Bayesian deciphering. We first show how the lexinvariant LM maintains uncertainty over possible lexical permutations while iteratively updating them at each index, using examples from a character-level lexinvariant model. Then, we also show an example of semantic in-context deciphering with a 32K vocabulary lexinvariant model, where the meaning of a novel word is inferred relative to common words in-context.

3.4.1 Uncertainty over Lexical Permutations

In Figure 6a, we input the following ciphered sequence to the frozen character-level lexinvariant LM with the probe: "I saw lots of people in town today, walking and talking around me. I greeted my friend Alice and my classmate Alex. I saw a guy, Joe, walking outside carrying a zat. Joe's zat was taken off zy wind. Today's wind was strong, so Joe's zat flew zackward. Joe lost Joe's zat for good. Joe will miss Joe's zat." For each instance of z in the sequence, we display the predicted deciphering of that instance as a row of probabilities across non-cipher letters.

The lexinvariant model starts off assuming uniform probability for all possible lexical permutations. After seeing more and more text, the lexinvariant model quickly realizes that z has a few main plausible decipherings: b, h, c, m. Eventually, the lexinvariant model is able to narrow the possibilities down to z maps to b near the end of the sequence. The predicted probabilities shift with the seen context accordingly, demonstrating an example of how the predicted cipher key is iteratively updated at each index.

Figure 6b shows another example with a similar set up, but with text "I saw a man in the pazk with a zat. The man was walking with the zat zight beside him. I've nevez seen anything like that befoze." While context initially suggests that z may be deciphered as c, it becomes clear that z must correspond to after the appearance of "right". The disambiguation is reflected in the depicted probabilities.

In Figure 6c and 6d, we show two deciphering examples over code. We consider two code examples in which it is initially ambiguous whether the character deciphers to or { . The ambiguity is eventually resolved by the use of Python-like or Java-like syntax.

3.4.2 Semantic Deciphering

In addition to character-level deciphering, we show examples of semantic deciphering with the larger vocabulary of 32k. Although the lexinvariant LM could not possibly gure out the true lexical permutation among 32k tokens using a small 512 context, it is possible to construct a simple context that repetitively uses simple words so that these words are easier to decipher. Then the lexinvariant LM can decipher the approximate semantics of the rare symbols relative to other easier-to-decipher words.

One example is the following: given the prompt 'crash!' 'aaah!' i looked up from my cup of coffee. 'crash!' - that was the cafe window. and 'aaah!' [... more text...] what one here is a drink ⏎ - restaurants ⏎ - music ⏎ - coffee ⏎ - father ⏎ the one here that drink is , where the word coffee , music, and father all only appear once before the question and restaurants appeared 4 times, the model is able to correctly answer that coffee is drinkable. See the full example in the appendix.

3.5 Synthetic Reasoning Tasks

As discussed in the introduction, lexical exibility is correlated with in-context reasoning performance, as demonstrated by existing large LMs. Thus, we study whether the lexinvariant model also learns in-context reasoning capabilities through the challenging lexinvariant training.

Speci cally, we measure the performance of lexinvariant models over two pure in-context symbol manipulation tasks: LookUp, where the task is to predict the next token based on the given lookup table, e.g. A->2 ⏎ C->4 ⏎ G->5 ⏎ C-> (should predict 4 here); and Permutation, where the task is to permute an arbitrary subsequence of the given sequence the same way as in the given few demonstrations, e.g. A 2 C->C , ⏎ 4 1 D-> (should predict D 4 here). In each of the tasks, the symbols are randomly sampled from the vocabulary so that we measure the pure reasoning ability independent from any knowledge of speci c words. We measure the model performance in terms of generated token accuracy over 1000 examples. The results are shown in Table 1. As shown in the table, the lexinvariant models achieve drastically higher accuracy, with an average of 4X improvement.

Table 1: Accuracy over synthetic reasoning tasks.

Dataset	Vocab	LookUp Acc		Permutation Acc	
		Standard	LI	Standard	LI
Pile	char	48.50	91.80	27.66	59.35
	32k	21.45	92.10	22.84	55.63
Wiki-40B	char	38.25	59.70	20.77	60.51
	32k	8.75	59.35	9.94	50.91
Github	char	42.40	86.65	21.03	71.59
	32k	4.25	80.20	8.59	67.39

(a) True deciphering: " ! "b", T = 1 .

(c) True deciphering: " ! "{", T = 2 .

(b) True deciphering: " ! "r", T = 1 .

(d) True deciphering: " ! ":", T = 3 .

Figure 6: Probe predictions for deciphering " at each occurrence of " in context.

3.6 Regularizing Language Models with Lexinvariance

Although lexinvariant LM has various interesting properties , it is not suitable for practical tasks since it would require the context to be extremely long so that all required words and knowledge are de ned in the context. Here, we explore how to construct more practical semi-lexinvariant LMs that maintain some properties of lexinvariant LMs via regularization. We emphasize that this exploration is intended to be illustrative rather than directly improving state-of-the-art.

Instead of using random Gaussian embedding matrices in place of a learned embedding matrix entirely, we can use random embeddings for only some of the tokens in each sequence, while others use the learned embedding. This means that the learned LM assumes that certain tokens have stable meanings but not others, which can be seen as a form of regularization towards lexinvariance. Speci cally, we randomly select tokens to randomize based on a Bernoulli distribution, which can essentially be seen as a form of dropout on token embeddings. On the BIG-bench tasks, we found that a model with dropout rate$p = 0:2$ for randomization was 25% more likely to improve performance than to harm performance when evaluated with three shots, relative to a comparably-sized LM, with improvements especially over retrieval type of tasks. See full details in the Appendix G.

More broadly, this regularization view could potentially bring the bene t of lexinvariant LMs to practical applications. For example, the regularization could improve 1) the robustness of LMs by making them less sensitive to adversarial attacks or noise in the input data, 2) generalization across different languages or domains by being less tied to speci c lexical items and more prone to learn the shared language structure, and 3) reasoning over more realistic tasks as we have started to explore with BIG-Bench. These areas are promising directions for future work to explore.

4 Related Work

4.1 Symbol Grounding

Beyond a modeling choice, the main question of our paper (that being whether an LM can learn language without a stable token representation) is also analogous to the symbol grounding problem: Can meaning be acquired when symbols are not even grounded stably, i.e. they can be mapped to completely random meanings in different sequences? It has long been argued by the symbol grounding literature that symbolic representations must be grounded bottom-up in nonsymbolic representations [21], with famous arguments like Searle's Chinese room: It describes a person in a room given a step-by-step set of instructions by which they can respond to Chinese text with reasonable-sounding Chinese text. To an outside observer, the person in the room appears to understand Chinese, but the individual does not know a word of Chinese. This is widely used to argue that understanding language requires grounding the symbols in the real world. It leads to an ongoing debate on whether LMs can learn meaning purely from large amounts of text, without grounding to any real-world objects [4]. Although intuitively, lexinvariant LMs appears one step further removed from physical grounding than standard LM, we nd that given enough context they can still infer the meaning of symbols based on lexical structures within the context.

4.2 Group invariances and Data augmentation

Our implementation of lexinvariant LMs can be seen as performing a form of very aggressive data augmentation, where we randomize the identity of each token in each sequence. From this perspective, it is somewhat similar to the data recombination in [14, 2] and augmentation of named entities in [20], where certain parts of the sentence are swapped with other words while still maintaining the original grammatical structure. In contrast to these augmentations, the training for our lexinvariant LMs completely swaps out all parts of the input text.

4.3 Byte-level T5

There is existing work on absorbing tokenization completely into part of language modeling by using extremely small tokens, such as Byte-level T5 [25]. In the extreme, such a model would become closer and closer to lexinvariant LM, since bytes or bits have almost no stable meaning, so their embeddings are likely not used for prediction. In this paper, we study general lexinvariant LMs with the lexinvariant property baked in and without requiring speci c tokenizers.

4.4 Deciphering Substitution Cipher using LMs

In general, solving substitution ciphers, where the cipher key is a permutation of the original alphabet, is a NP-hard problem when only having access to LMs that can assign probabilities to sequences [18]. There have been several works focusing on solving substitution ciphers using LMs, including approaches from searching over the permutation space guided by LMs' scores [12] to training a seq-to-seq model directly to perform deciphering as translation [8]. Although our work does not focus specifically on the task of deciphering substitution ciphers, we show that our lexinvariant model can efficiently perform in-context deciphering as a byproduct of language modeling.

4.5 Reasoning

It has been shown that large language models acquire surprising in-context reasoning capabilities [6, 15, 23]. Many of them are related to lexical flexibility through training for purely next-token prediction, such as modified arithmetic, data reformatting, and redefining single word etc. However, LLMs also memorize an enormous amount of knowledge along the way, which is often unnecessary. This work can also be seen as an exploration of whether a (semi-)lexinvariant LM can discount knowledge and prioritize learning the diverse structural reasoning patterns in language, therefore achieving the strong reasoning capability of LLMs with a smaller model.

5 Conclusion

In this work, we define and study lexinvariant language models, which do not have stable embeddings and learn to infer the meaning of symbols in-context. We show several surprising properties of this model theoretically and empirically, including convergence to standard language modeling, in-context deciphering, and better reasoning capabilities. We also explore a less extreme lexinvariance regularized language model and demonstrate its potential for solving more practical tasks efficiently.

Acknowledgments and Disclosure of Funding

We thank Sang Michael Xie and Steven Cao for discussions and for providing feedback on our manuscript. This project is supported by Open Philanthropy Project Award. Qian Huang is supported by Open Philanthropy AI fellowship.

References

[1] The multiplicative weights algorithm. https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture16.pdf. Accessed: 2023-04-24.

[2] E. Akyürek, A. F. Akyurek, and J. Andreas. Learning to recombine and resample data for compositional generalization. ArXiv, abs/2010.03706, 2020.

[3] N. Aldarrab and J. May. Can sequence-to-sequence models crack substitution ciphers? In Annual Meeting of the Association for Computational Linguistics, 2020.

[4] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, 2021.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. ArXiv, abs/2005.14165, 2020.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv, abs/1810.04805, 2019.

[8] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In European Conference on Computational Learning Theory, 1997.

[9] L. Gao, S. R. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy. The pile: An 800gb dataset of diverse text for language modeling. ArXiv, abs/2101.00027, 2020.

[10] M. Guo, Z. Dai, D. Vrandecic, and R. Al-Rfou. Wiki-40b: Multilingual language model dataset. In LREC 2020, 2020.

[11] S. Harnad. Symbol grounding problem. Scholarpedia, 2:2373, 1990.

[12] B. Hauer, R. B. Hayward, and G. Kondrak. Solving substitution ciphers with combined language models. In International Conference on Computational Linguistics, 2014.

[13] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models. ArXiv, abs/2203.15556, 2022.

[14] R. Jia and P. Liang. Data recombination for neural semantic parsing. ArXiv, abs/1606.03622, 2016.

[15] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. R'e, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. J. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. S. Kim, N. Guha, N. S. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. F. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda. Holistic evaluation of language models. ArXiv, abs/2211.09110, 2022.

[16] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Ef cient estimation of word representations in vector space. In International Conference on Learning Representations, 2013.

[17] T. Mikolov, M. Kara át, L. Burget, J. Cernock, and S. Khudanpur. Recurrent neural network based language model. In Interspeech, 2010.

[18] M. Nuhn and H. Ney. Decipherment complexity in 1:1 substitution ciphers. In Annual Meeting of the Association for Computational Linguistics, 2013.

[19] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a uni ed text-to-text transformer. J. Mach. Learn. Res., 21(1), jun 2022.

[20] J. Raiman and J. Miller. Globally normalized reader. Conference on Empirical Methods in Natural Language Processing, 2017.

[21] H. Schütze. Part-of-speech induction from scratch. 31st Annual Meeting of the Association for Computational Linguistics, pages 251–258, Columbus, Ohio, USA, June 1993. Association for Computational Linguistics.

[22] N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In J. G. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of Proceedings of Machine Learning Research, pages 4603–4611. PMLR, 2018.

[23] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, A. Kluska, A. Lewkowycz, A. Agarwal, A. Power, A. Ray, A. Warstadt, A. W. Kocurek, A. Safaya, A. Tazarv, A. Xiang, A. Parrish, A. Nie, A. Hussain, A. Askell, A. Dsouza, A. A. Rahane, A. S. Iyer, A. Andreassen, A. Santilli, A. Stuhlmuller, A. M. Dai, A. D. La, A. K. Lampinen, A. Zou, A. Jiang, A. Chen, A. Vuong, A. Gupta, A. Gottardi, A. Norelli, A. Venkatesh, A. Gholamidavoodi, A. Tabassum, A. Menezes, A. Kirubarajan, A. Mullokandov, A. Sabharwal, A. Herrick, A. Efrat, A. Erdem, A. Karakacs, B. R. Roberts, B. S. Loe, B. Zoph, B. Bojanowski, B. Ozyurt, B. Hedayatnia, B. Neyshabur, B. Inden, B. Stein, B. Ekmekci, B. Y. Lin, B. S. Howald, C. Diao, C. Dour, C. Stinson, C. Argueta, C. F. Ram'irez, C. Singh, C. Rathkopf, C. Meng, C. Baral, C. Wu, C. Callison-Burch, C. Waites, C. Voigt, C. D. Manning, C. Potts, C. T. Ramirez, C. Rivera, C. Siro, C. Raffel, C. Ashcraft, C. Garbacea, D. Sileo, D. H. Garrette, D. Hendrycks, D. Kilman, D. Roth, D. Freeman, D. Khashabi, D. Levy, D. Gonz'alez, D. Hernandez, D. Chen, D. Ippolito, D. Gilboa, D. Dohan, D. Drakard, D. Jurgens, D. Datta, D. Ganguli, D. Emelin, D. Kleyko, D. Yuret, D. Chen, D. Tam, D. Hupkes, D. Misra, D. Buzan, D. C. Mollo, D. Yang, D.-H. Lee, E. Shutova, E. D. Cubuk, E. Segal, E. Hagerman, E. Barnes, E. P. Donoway, E. Pavlick, E. Rodolà, E. F. Lam, E. Chu, E. Tang, E. Erdem, E. Chang, E. A. Chi, E. Dyer, E. J. Jerzak, E. Kim, E. E. Manyasi, E. Zheltonozhskii, F. Xia, F. Siar, F. Mart'inez-Plumed, F. Happ'e, F. Chollet, F. Rong, G. Mishra, G. I. Winata, G. de Melo, G. Kruszewski, G. Parascandolo, G. Mariani, G. Wang, G. Jaimovitch-L'opez, G. Betz, G. Gur-Ari, H. Galijasevic, H. S. Kim, H. Rashkin, H. Hajishirzi, H. Mehta, H. Bogar, H. Shevlin, H. Schütze, H. Yakura, H. Zhang, H. Wong, I. A.-S. Ng, I. Noble, J. Jumelet, J. Geissinger, J. Kernion, J. Hilton, J. Lee, J. F. Fisac, J. B. Simon, J. Koppel, J. Zheng, J. Zou, J. Koco'n, J. Thompson, J. Kaplan, J. Radom, J. N. Sohl-Dickstein, J. Phang, J. Wei, J. Yosinski, J. Novikova, J. Bosscher, J. Marsh, J. Kim, J. Taal, J. Engel, J. O. Alabi, J. Xu, J. Song, J. Tang, J. W. Waweru, J. Burden, J. Miller, J. U. Balis, J. Berant, J. Frohberg, J. Rozen, J. Hernández-Orallo, J. Boudeman, J. Jones, J. B. Tenenbaum, J. S. Rule, J. Chua, K. Kanclerz, K. Livescu, K. Krauth, K. Gopalakrishnan, K. Ignatyeva, K. Markert, K. D. Dhole, K. Gimpel, K. O. Omondi, K. W. Mathewson, K. Chiafullo, K. Shkaruta, K. Shridhar, K. McDonell, K. Richardson, L. Reynolds, L. Gao, L. Zhang, L. Dugan, L. Qin, L. Contreras-Ochando, L.-P. Morency, L. Moschella, L. Lam, L. Noble, L. Schmidt, L. He, L. O. Col'on, L. Metz, L. K. cSenel, M. Bosma, M. Sap, M. ter Hoeve, M. Andrea, M. S. Farooqi, M. Faruqui, M. Mazeika, M. Baturan, M. Marelli, M. Maru, M. Quintana, M. Tolkiehn, M. Giulianelli, M. Lewis, M. Potthast, M. Leavitt, M. Hagen, M. Schubert, M. Baitemirova, M. Arnaud, M. A. McElrath, M. A. Yee, M. Cohen, M. Gu, M. I. Ivanitskiy, M. Starritt, M. Strube, M. Swkedrowski, M. Bevilacqua, M. Yasunaga, M. Kale, M. Cain, M. Xu, M. Suzgun, M. Tiwari, M. Bansal, M. Aminnaseri, M. Geva, M. Gheini, T. MukundVarma, N. Peng, N. Chi, N. Lee, N. G.-A. Krakover, N. Cameron, N. S. Roberts, N. Doiron, N. Nangia, N. Deckers, N. Muennighoff, N. S. Keskar, N. Iyer, N. Constant, N. Fiedel, N. Wen, O. Zhang, O. Agha, O. Elbaghdadi, O. Levy, O. Evans, P. A. M. Casares, P. Doshi, P. Fung, P. P. Liang, P. Vicol, P. Alipoormolabashi, P. Liao, P. Liang, P. W. Chang, P. Eckersley, P. M. Htut, P.-B. Hwang, P. Milkowski, P. S. Patil, P. Pezeshkpour, P. Oli, Q. Mei, Q. LYU, Q. Chen, R. Banjade, R. E. Rudolph, R. Gabriel, R. Habacker, R. R. Delgado, R. Millière, R. Garg, R. Barnes, R. A. Saurous, R. Arakawa, R. Raymaekers, R. Frank, R. Sikand, R. Novak, R. Sitelew, R. L. Bras, R. Liu, R. Jacobs, R. Zhang, R. Salakhutdinov, R. Chi, R. Lee, R. Stovall,

R. Teehan, R. Yang, S. J. Singh, S. M. Mohammad, S. Anand, S. Dillavou, S. Shleifer, S. Wiseman, S. Gruetter, S. Bowman, S. S. Schoenholz, S. Han, S. Kwatra, S. A. Rous, S. Ghazarian, S. Ghosh, S. Casey, S. Bischoff, S. Gehrmann, S. Schuster, S. Sadeghi, S. S. Hamdan, S. Zhou, S. Srivastava, S. Shi, S. Singh, S. Asaadi, S. S. Gu, S. Pachchigar, S. Toshniwal, S. Upadhyay, S. Debnath, S. Shakeri, S. Thormeyer, S. Melzi, S. Reddy, S. P. Makini, S. hwan Lee, S. B. Torene, S. Hatwar, S. Dehaene, S. Divic, S. Ermon, S. R. Biderman, S. C. Lin, S. Prasad, S. T. Piantadosi, S. M. Shieber, S. Misherghi, S. Kiritchenko, S. Mishra, T. Linzen, T. Schuster, T. Li, T. Yu, T. A. Ali, T. Hashimoto, T.-L. Wu, T. Desbordes, T. Rothschild, T. Phan, T. Wang, T. Nkinyili, T. Schick, T. N. Kornev, T. Telleen-Lawton, T. Tunduny, T. Gerstenberg, T. Chang, T. Neeraj, T. Khot, T. O. Shultz, U. Shaham, V. Misra, V. Demberg, V. Nyamai, V. Raunak, V. V. Ramasesh, V. U. Prabhu, V. Padmakumar, V. Srikumar, W. Fedus, W. Saunders, W. Zhang, W. Vossen, X. Ren, X. F. Tong, X. Wu, X. Shen, Y. Yaghoobzadeh, Y. Lakretz, Y. Song, Y. Bahri, Y. J. Choi, Y. Yang, Y. Hao, Y. Chen, Y. Belinkov, Y. Hou, Y. Hou, Y. Bai, Z. Seid, Z. Xinran, Z. Zhao, Z. F. Wang, Z. J. Wang, Z. Wang, Z. Wu, S. Singh, and U. Shaham. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *ArXiv*, abs/2206.04615, 2022.

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

[25] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2021.

A Convergence Proof

Theorem A.1. Let $x_1, \ldots, x_n$ be any token sequence generated by an arbitrary language distribution p with an alphabet of size d. Let $p'(x_1, \ldots, x_n) = E_\pi[p(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n))]$. Then, for any $0 < \varepsilon < 1/2$,

$$\frac{1}{T}\sum_{t=1}^{T} \| p(x_t \mid x_1, \ldots, x_{t-1}) - p'(x_t \mid x_1, \ldots, x_{t-1}) \|_1$$

with probability greater than $1 - \delta$ when $T \geq \frac{d}{\varepsilon^4}\,\mathrm{polylog}(d, \frac{1}{\varepsilon}, \frac{1}{\delta})$.

Proof. For any desired error $0 < \varepsilon < 1/2$ and failure rate $0 < \delta < 1/2$, we will first prove the analogous statement for KL divergence instead of L_1 distance, and then relate a bound on KL divergence back to L_1 distance via Pinsker's inequality.

Throughout the rest of proof, we will work with a parameter $0 < \alpha < O\left(\frac{\varepsilon}{(\log(1/\delta))^{1/4}}\right) < \frac{1}{2}$, and will bound our KL divergence by α^2.

To prove the bound in terms of KL divergence, it will be useful to ensure to work with a "smoothed" version of p, which we denote by $\tilde{p}$, for which every token has some nonzero probability, γ/d, of appearing at each timestep, for a parameter $\gamma = \alpha^2/T$:

$$\tilde{p}(x_T \mid x_1, \ldots, x_{T-1}) = p(x_T \mid x_1, \ldots, x_{T-1})(1 - \gamma) + \frac{\gamma}{d}.$$

Similarly, let $p'(x_1, \ldots, x_n) = E_\pi[\tilde{p}(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n))]$. We use $\tilde{P}'$ to denote the probabilities under this change. With probability at least $1 - \delta/2$, the realized sequence $x_1, \ldots, x_n$ drawn under $\tilde{p}$ can be regarded as being drawn from $\tilde{p}'$ (as these distributions can be coupled with this probability).

The key idea is then to show that $\tilde{p}'(y_{t+1} \mid y_{1:t})$, where $y_t = \pi(x_t)$ for some ground truth unknown to p', is equivalent to using the multiplicative weights algorithm to predict y_{t+1} with the Hedge strategy, with the experts being each possible permutation of the tokens and the cost incurred by each expert being the negative log likelihood of the prediction. We denote $\tilde{P}'_\pi(y_{1:n}) = \tilde{P}'(y_{1:n} \mid \pi = \pi) = \tilde{p}(\pi^{-1}(y_1), \ldots, \pi^{-1}(y_n))$ and show this in Lemma A.2.

With this equivalence, we can then bound the difference between the prediction and p' as the regret of the multiplicative weights algorithm. Concretely, we show in Lemma A.3 that the regret to p' to any expert π is bounded as

$$\frac{1}{T}\sum_{t} \log \frac{\tilde{P}'_\pi(y_{t+1} \mid y_{1:t})}{\tilde{p}'(y_{t+1} \mid y_{1:t})} \leq 2\alpha^2$$

for $T \geq 4\log^2(\frac{d}{\delta})\log(d!) = \alpha^4$.

We can see π as the particular expert/permutation. And we can further only consider the special case that π is also the identity permutation, then the same result holds for x_t and with $\tilde{P}'_\pi$ replaced by $\tilde{p}'$, i.e.

$$\frac{1}{T}\sum_{t} \log \frac{\tilde{p}(x_{t+1} \mid x_{1:t})}{\tilde{p}'(x_{t+1} \mid x_{1:t})} \leq 2\alpha^2$$

Now we want to convert this bound on regret in terms of log likelihood to KL divergence, and eventually to L_1 distance. To convert it to KL divergence regret, we construct a martingale:

$$Z_i = \sum_{t=1}^{i} D_{KL}(\tilde{p}(x_{t+1} \mid x_{1:t}) \| \tilde{p}'(x_{t+1} \mid x_{1:t})) - \log \frac{\tilde{p}(x_{t+1} \mid x_{1:t})}{\tilde{p}'(x_{t+1} \mid x_{1:t})}.$$

We verify that this is a martingale in Lemma A.4, with differences bounded by $2\log\frac{1}{\gamma}$, and bound the probability that Z_T exceeds $b = \log\frac{d}{\gamma}\sqrt{8T\log\frac{2}{\delta}}$ via Azuma's inequality Lemma A.6: with probability $1 - \delta/2$, we have that $|Z_T| \leq b$.

Therefore, we have that with probability at least $1 - \delta/2$

$$Z_T = \sum_{t=1}^{T} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right) - \log \frac{p(x_{t+1} \mid x_{1:t})}{p^0(x_{t+1} \mid x_{1:t})} \le b$$

$$\sum_{t=1}^{T} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right) \le \sum_{t=1}^{T} \log \frac{p(x_{t+1} \mid x_{1:t})}{p^0(x_{t+1} \mid x_{1:t})} + b$$

Putting this all together, since $\frac{1}{T}\sum_t^T \log \frac{p_t(x_{t+1} \mid x_{1:t})}{p^0(x_{t+1} \mid x_{1:t})} \le \alpha^2$ for $T \ge 4\log^2(\frac{d}{\delta})\log(d!)/\alpha^4$, we have the following:

$$\sum_{1}^{T} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right) \le \alpha^2 T + b.$$

We now convert our bound on KL divergence to a bound on L_1 distance via Pinsker's inequality:

$$\|p(x_{t+1} \mid x_{1:t}) - p^0(x_{t+1} \mid x_{1:t})\|_1 \le \sqrt{\frac{1}{2} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right)}.$$

Further, at any given x_t, the difference between the redistributed probability distribution p and a unmodified probability distribution p is at most γ, so

$$\|p(x_{t+1} \mid x_{1:t}) - p^0(x_{t+1} \mid x_{1:t})\|_1 \le \|p(x_{t+1} \mid x_{1:t}) - p^0(x_{t+1} \mid x_{1:t})\|_1 + 2\gamma.$$

We are interested in the average L_1 across time steps:

$$\frac{1}{T}\sum_{t=1}^{T} \|p(x_{t+1} \mid x_{1:t}) - p^0(x_{t+1} \mid x_{1:t})\|_1 \le \frac{1}{T}\sum_{t=1}^{T} \left(\|p(x_{t+1} \mid x_{1:t}) - p^0(x_{t+1} \mid x_{1:t})\|_1 + 2\gamma\right)$$

$$\le \frac{1}{T}\sum_{t=1}^{T} \sqrt{\frac{1}{2} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right)} + 2\gamma$$

$$\le \sqrt{\frac{1}{T}\sum_{t=1}^{T} \frac{1}{2} D_{KL}\left(p(x_{t+1} \mid x_{1:t}) \| p^0(x_{t+1} \mid x_{1:t})\right)} + 2\gamma,$$

where in the last inequality we applied Cauchy–Schwarz. Hence for $T \ge 4\log^2\frac{d}{\delta}\log(d!)/\alpha^4$,

$$\frac{1}{T}\sum_{t=1}^{T} \|p(x_{t+1} \mid x_1, \ldots, x_t) - p^0(x_{t+1} \mid x_1, \ldots, x_t)\|_1 \le \sqrt{\frac{1}{T}\left(\frac{T}{2}(2\alpha^2 T + b)\right)} + 2\gamma$$

$$\le \sqrt{\alpha^2 + \frac{b}{2T}} + 2\gamma.$$

Simplifying this for $b = \log\frac{d}{\delta}\sqrt{8T\log\frac{2}{\gamma}}$, $T \ge 4\log^2(\frac{d}{\delta})\log(d!)/\alpha^4$ and $\gamma = \alpha^2/T$, we have

$$\frac{1}{T}\sum_{t=1}^{T} \|p(x_{t+1} \mid x_1, \ldots, x_t) - p^0(x_{t+1} \mid x_1, \ldots, x_t)\|_1 \le \sqrt{\alpha^2 + \sqrt{\frac{2\log\frac{2}{\gamma}}{\log(d!)}}\,\alpha^2 + \frac{2\alpha^2}{T}}$$

$$\le \alpha\left(\sqrt{\frac{2}{T}} + \sqrt{1 + \sqrt{\frac{2\log\frac{2}{\gamma}}{\log(d!)}}}\right)$$

$$\le \alpha\left(1 + \sqrt{1 + \sqrt{2\log\tfrac{2}{\gamma}}}\right) \le 2\sqrt{2}(2\log\tfrac{2}{\gamma})^{1/4}\alpha.$$

We can bound this average error by ϵ if we set $\epsilon_0 = \frac{\epsilon}{2^{\frac{1}{2}(2\log\frac{2}{\epsilon})^{1/4}}} < \frac{1}{2}$, in which case our condition that $T \geq 4\log^2(\frac{dT}{\epsilon_0})\log(d!) = \epsilon^4$ becomes $T \geq 512\log\frac{2}{\epsilon}\log^2\frac{dT}{\epsilon_0}\log(d!) = \epsilon^4$. The theorem now follows by simplifying this expression. Since $\log\frac{2}{\epsilon} \leq 2\log\frac{1}{\epsilon}$, and $\log(d!) \leq d\log(d)$, we can relax the condition on T as

$$T \geq 1024\log\frac{1}{\epsilon}\log^2(\frac{d}{\epsilon_0})\log^2(T)d\log(d) = \epsilon^4 = \log^2(T)\frac{d}{\epsilon^4}\text{polylog}(d, \frac{1}{\epsilon}, \frac{1}{\epsilon})$$

To remove the $\log^2 T$ from the right side, note that for any $W > 10$, if $T > 10\, W \log^2 W$, then $T > W\log^2 T$, yielding the further relaxed the condition on T as

$$T \geq \frac{d}{\epsilon^4}\text{polylog}(d, \frac{1}{\epsilon}, \frac{1}{\epsilon}).$$

□

Lemma A.2. Consider an arbitrary ground truth permutation σ^*. For all time steps $t \in [1, n]$, let $y_t = \sigma^*(x_t)$. Consider the online prediction game of predicting y_{t+1} at each time step given previous observation $y_{1:t}$ without knowing σ^* but knowing $\mathcal{D}$. Then $\Pr^{\sigma^*}(y_{t+1} \mid y_{1:t})$ is equivalent to the multiplicative weights algorithm's prediction of y_{t+1} with the Hedge strategy of Freund and Schapire [8], where it

- Considers $d!$ experts corresponding to guessing each permutation σ is the ground truth permutation.
- Maintains a weight $w_\sigma^{(t)}$ for each expert at time step t, and the weights are initially all $\Pr(\sigma)$.
- Picks a distribution across experts $p_\sigma^{(t)} = \frac{w_\sigma^{(t)}}{\Phi^{(t)}}$ where $\Phi^{(t)} = \sum_j w_j^{(t)}$.
- Produces prediction of y_{t+1} as $\sum_\sigma p_\sigma^{(t)} \Pr^\sigma(y_{t+1} \mid y_{1:t})$
- Receives a cost vector of $m_\sigma^{(t)} = -\frac{1}{\epsilon}\log\Pr^\sigma(y_{t+1} \mid y_{1:t})$.
- Updates the weights as $w_i^{(t+1)} = w_i^{(t)}\exp(-\epsilon m_i^{(t)})$ and repeat

Proof. We can first see that $p_\sigma^{(t)} = \Pr(\sigma \mid y_{1:t})$ by induction:

Base case $p_\sigma^{(0)} = \Pr(\sigma)$ by assumption.

Inductive Case:

With the cost vector as $m_\sigma^{(t-1)} = -\frac{1}{\epsilon}\log\Pr^\sigma(y_t \mid y_{1:t-1})$, the update at step t is $w_\sigma^{(t)} = w_\sigma^{(t-1)}\Pr^\sigma(y_t \mid y_{1:t-1})$. Therefore, the probability over any particular expert is

$$\begin{aligned}
p_\sigma^{(t)} &= \frac{w_\sigma^{(t)}}{\Phi^{(t)}} \\
&= \frac{w_\sigma^{(t-1)}\Pr^\sigma(y_t \mid y_{1:t-1})}{\sum_j w_j^{(t-1)}\Pr^j(y_t \mid y_{1:t-1})} \\
&= \frac{p_\sigma^{(t-1)}\Phi^{(t-1)}\Pr^\sigma(y_t \mid y_{1:t-1})}{\sum_j p_j^{(t-1)}\Phi^{(t-1)}\Pr^j(y_t \mid y_{1:t-1})} \\
&= \frac{p_\sigma^{(t-1)}\Pr^\sigma(y_t \mid y_{1:t-1})}{\sum_j p_j^{(t-1)}\Pr^j(y_t \mid y_{1:t-1})}
\end{aligned}$$

This is equivalent to the update given by Bayes' rule when plugging in $p_\sigma^{(t)} = \Pr(\sigma \mid y_{1:t})$:

$$\Pr(\sigma \mid y_{1:t}) = \frac{\Pr(\sigma \mid y_{1:t-1})\Pr^\sigma(y_t \mid y_{1:t-1})}{\Pr(y_t \mid y_{1:t-1})}$$

So we can conclude that $p_{\sigma}^{(t)} = \Pr(\sigma \mid y_{1:t})$, i.e. the process of updating the probability distribution across experts within the prediction game is equivalent to the process of the language model updating the probabilities $\Pr(\sigma \mid y_{1:t+1})$ across permutations σ. And this means that the algorithm's prediction

$$\sum_{\sigma} p_{\sigma}^{(t)} \Pr_{\sigma}(y_{t+1} \mid y_{1:t}) = \sum_{\sigma} \Pr(\sigma \mid y_{1:t}) \Pr_{\sigma}(y_{t+1} \mid y_{1:t}) = \Pr(y_{t+1} \mid y_{1:t}) = p^{\sigma}(y_{t+1} \mid y_{1:t}) \qquad \square$$

Lemma A.3. When using the Hedge strategy for the multiplicative weights algorithm, the average difference between the weighted distribution across experts and any particular expert is bounded as

$$\frac{1}{T} \sum_{t} \log \frac{\Pr(y_{t+1} \mid y_{1:t})}{p^{\sigma}(y_{t+1} \mid y_{1:t})} \quad 2\,^{2}$$

for 1 and for $T \quad 4\log^{2} \frac{d}{} \log(d!) = {}^{4}$.

Proof. Consider an arbitrary expert

We first show that the cost vectors are bounded by $= \frac{1}{} \log \frac{}{d}$: Recall we defined $m^{(t)} = \frac{1}{} \log \Pr(y_{t+1} \mid y_{1:t})$. By the definition of our redistributed probability distribution, at time step $t \in [1, \ldots, T]$,

$$\begin{aligned} \frac{}{d} \quad \Pr(y_{t+1} \mid y_{1:t}) \quad 1 \\ \log \frac{}{d} \quad \log \Pr(y_{t+1} \mid y_{1:t}) \quad 0 \\ 0 \quad m^{(t)} \quad \frac{1}{} \log \frac{}{d} \\ 0 \quad m^{(t)} \quad \frac{1}{} \log \frac{}{d}. \end{aligned}$$

By corollary 16.3 in [1], if we have cost vectors $m^{(t)} \in [\ ;\ ]^{d!}$, then for time $T \quad (4\ ^{2}\log(d!)) = {}^{2}$ where 1,

$$\frac{1}{T} \sum_{t} p^{(t)} \cdot m^{(t)} \quad \frac{1}{T} \sum_{t} m^{(t)} + 2\ .$$

Note that we can simplify $T \quad 4\log^{2} \frac{d}{} \log(d!) = {}^{4}$.

We can now bound

$$\begin{aligned} \frac{1}{T} \sum_{t} p^{(t)} \cdot m^{(t)} \quad m^{(t)} \quad 2 \\ \left(\frac{1}{T} \sum_{t} \sum_{\sigma} p_{\sigma}^{(t)} m_{\sigma}^{(t)} \quad m^{(t)} \quad 2 \right) \\ \left(\frac{1}{T} \sum_{t} \sum_{\sigma} \Pr(\sigma \mid y_{1:t}) \quad \frac{1}{} \log \Pr_{\sigma}(y_{t+1} \mid y_{1:t}) \quad \frac{1}{} \log \Pr(y_{t+1} \mid y_{1:t}) \quad 2 \right) \\ \frac{1}{T} \sum_{t} \sum_{\sigma} \Pr(\sigma \mid y_{1:t}) \log \Pr(y_{t+1} \mid y_{1:t}) \log \Pr_{\sigma}(y_{t+1} \mid y_{1:t}) \quad 2 \\ \frac{1}{T} \sum_{t} \mathrm{E}_{\sigma} \log \frac{\Pr(y_{t+1} \mid y_{1:t})}{\Pr_{\sigma}(y_{t+1} \mid y_{1:t})} \quad 2\,^{2} \end{aligned}$$

By Jensen's inequality, we also have that

$$\begin{aligned} \frac{1}{T} \sum_{t} \log \frac{\Pr(y_{t+1} \mid y_{1:t})}{\mathrm{E}_{\sigma} \Pr_{\sigma}(y_{t+1} \mid y_{1:t})} \quad 2\,^{2} \\ \frac{1}{T} \sum_{t} \log \frac{\Pr(y_{t+1} \mid y_{1:t})}{p^{\sigma}(y_{t+1} \mid y_{1:t})} \quad 2\,^{2} \end{aligned}$$

□

Lemma A.4. *Let*

$$Z_i = \sum_{t=1}^{\dot{X}} D_{KL}(P_l(x_{t+1}|x_{1:t}) \| P(x_{t+1}|x_{1:t})) \quad \log \frac{P_l(x_{t+1}|x_{1:t})}{P(x_{t+1}|x_{1:t})}$$

Z_i is a martingale.

Proof. Consider

$$\begin{aligned} \mathsf{E}_{x_{i+1} \quad \tilde{P}_l}[Z_i] &= \mathsf{E}_{x_{i+1} \quad \tilde{P}_l} \left[\sum_{t=1}^{\dot{X}} D_{KL}(P_l(x_{t+1}|x_{1:t}) \| P(x_{t+1}|x_{1:t})) \quad \log \frac{P_l(x_{t+1}|x_{1:t})}{P(x_{t+1}|x_{1:t})} \right] \\ &= \mathsf{E}_{x_{i+1} \quad \tilde{P}_l} \left[D_{KL}(P_l(x_{i+1}|x_{1:i}) \| P(x_{i+1}|x_{1:i})) \quad \log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} + Z_{i \quad 1} \right] \end{aligned}$$

Observe that $Z_{i \quad 1}$ has no dependence on x_{i+1}.

$$\begin{aligned} \mathsf{E}_{x_{i+1} \quad \tilde{P}_l}[Z_i] &= \mathsf{E}_{x_{i+1} \quad \tilde{P}_l} \left[\mathsf{E}_{x_{i+1} \quad \tilde{P}_l} \log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} \right] \quad \mathsf{E}_{x_{i+1} \quad \tilde{P}_l} \left[\log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} \right] + Z_{i \quad 1} \\ &= Z_{i \quad 1} \end{aligned}$$

Therefore, Z_i is a martingale. □

Lemma A.5. $|Z_i \quad Z_{i \quad 1}| \quad c_i$ *where* $c_i = 2|\log \frac{d}{}|$

Proof. We have

$$|Z_i \quad Z_{i \quad 1}| = \left| D_{KL}(P_l(x_{i+1}|x_{1:i}) \| P(x_{i+1}|x_{1:i})) \quad \log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} \right|$$

In our redistributed probability distribution P, we have $\frac{}{d} \quad P \ (x_i|x_{1:i \quad 1}) \quad 1$ for any $\quad$ at any time i. Therefore,

$$\log \frac{}{d} \quad \log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} \quad \log \frac{d}{}:$$

Also, we can find an upper bound for the KL divergence by maximizing $P_l(x_{i+1}|x_{1:i})$ to 1 and minimizing $P(x_{i+1}|x_{1:i})$ to $\frac{}{d}$ so that

$$D_{KL}(P_l(x_{i+1}|x_{1:i}) \| P(x_{i+1}|x_{1:i})) = \sum_{x_{i+1}} P_l(x_{i+1}|x_{1:i}) \log \frac{P_l(x_{i+1}|x_{1:i})}{P(x_{i+1}|x_{1:i})} \quad \log \frac{d}{}$$

We can maximize $|Z_i \quad Z_{i \quad 1}|$ by maximizing the first term and minimizing the second term, or vice versa. In the first case, $|Z_i \quad Z_{i \quad 1}| \quad |\log \frac{d}{} \quad \log \frac{}{d}| = 2|\log \frac{d}{}|$. In the other case, $|Z_i \quad Z_{i \quad 1}| \quad |0 \quad \log \frac{d}{}| = |\log \frac{d}{}|$.

Therefore, $|Z_i \quad Z_{i \quad 1}| \quad c_i$ where $c_i = 2|\log \frac{d}{}|$. □

Lemma A.6. *By Azuma's inequality, with probability* $1 \quad$ *, we have that* $\|Z_T\| \quad b$ *where* $b = 2\log \frac{d}{} \sqrt{8T \log \frac{1}{}}$

Proof. By Azuma's inequality, for all positive reals b,

$$P(Z_T \quad Z_1 \quad b) \quad \exp\left(\frac{b^2}{2\sum_{k=2}^{T} c_k^2}\right)$$

$$P(Z_T \quad Z_1 \quad b) \quad 1 \quad \exp\left(\frac{b^2}{2\sum_{k=2}^{T} c_k^2}\right)$$

$$1 \quad \exp\left(\frac{b^2}{8\sum_{k=2}^{T} \log^2 \frac{d}{}}\right)$$

We can rewrite in terms of $= \exp\left(\frac{b^2}{8\sum_{k=2}^{T} \log^2 \frac{d}{}}\right)$ so

$$b = \sqrt{8 \sum \log^2 \frac{d}{}} \log\left(\frac{d}{}\right) \sqrt{\frac{k=2}{8T \log \frac{1}{}}}$$

Therefore,

$$P(Z_T \quad Z_1 \quad b) \quad 1$$

□

B Model Architecture Details

In addition, we add a learnable scaling and bias parameter to the result of the embedding layer, so that the model can still learn to scale it as needed.

C Convergence on other datasets

Figure 7 shows the perplexity of lexinvariant LMs across the three different datasets. Note that Github converges significantly faster than standard Engish text like Wiki-40B, since code is more structured and easier to decipher the token permutation.

D Code Deciphering Full Examples

Java:

```
binary_search()z
  if (high >= low)z
    mid = (high + low) / 2;
    if (arr[mid] == x)
      return mid;
    if (arr[mid] > x)z
      high = mid - 1;
      return binary_search();
    } elsez
      low = mid + 1;
      return binary_search();
    }
  } elsez
    return -1;
  }
}
void func2()z
```

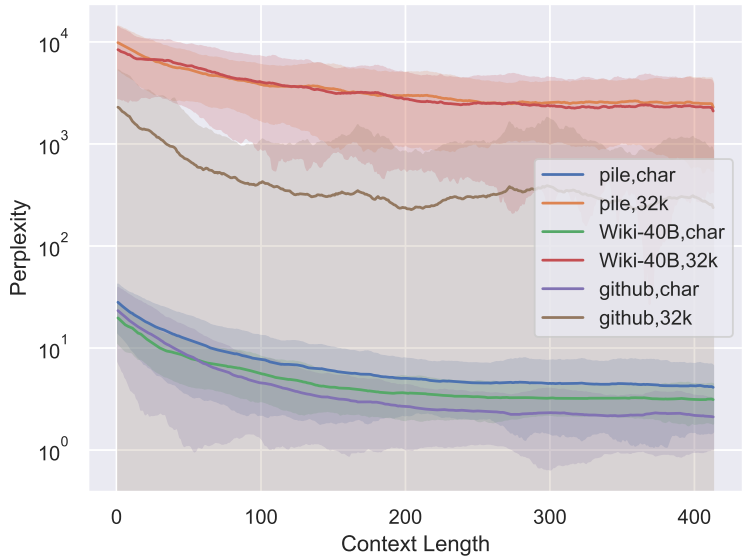


Figure 7: Smoothed Token Perplexity over the Pile, Wiki-40B and Github, with character-level and T5 default vocab

Python:

```
binary_search()z
  if (high >= low)z
    mid = (high + low) // 2
    if (arr[mid] == x)z
      return mid
    if (arr[mid] > x)z
      high = mid - 1
      return binary_search()
    elsez
      low = mid + 1
      return binary_search()
  elsez
      return -1
def func2()z
```

E Semantic Deciphering Full Example

```
'crash!' 'aaah!' i looked up from my cup of coffee. 'crash!' - that was
the cafe window. and 'aaah!' - that was kate. people in the cafe shouted.
kate and i ran to the window. there was no one there. then i turned to kate
and put my arm around her. 'are you all right?' i asked. 'yes,' she said.
'i think so.' 'what is it?' some one shouted and a short red-faced man ran
into the room. the man took my arm. 'matt! what are you doing to kate?'
he asked. 'nothing, papa,' kate replied. 'it wasn't him. it was from out
in the street.' the red-faced man looked at the window and then at me. he
turned to his daughter. 'are you ok, kate?' he asked. kate gave him a
little smile. 'yes, i think i am, papa,' she said. then her father spoke
to me. 'sorry, matt. i heard kate and i thought...' 'that's ok, paolo,' i
answered. it was ok. you see, this is soho, in the centre of london. in the
day it's famous for music and films. at night people come and eat and drink
```

```
in the restaurants.  expensive restaurants and cheap restaurants; italian
restaurants and chinese restaurants.  and day and night there are internet
cafes like the web cafe.  in soho you can buy any thing and any one.  there
are lots of nice people in soho.  but there are also lots of people who are
not very nice.  i know because i live and work here.  i often take a drink to
a shop or cafe.  i'm not rich and famous.  and i don't know a lot.  but i do
know soho.  what one here is a drink - restaurants - music - coffee - father
the one here that drink is
```

Example prediction of the lexinvariant with 32k vocabulary train on the Pile:

```
 - coffee.  and i
```

The probability (at temperature=1) of coffee being selected is 56%, substantially higher than the next-highest probability of restaurant at 27%, music at 12%, or father at 5%.

F Synthetic Reasoning Task

Table 2 shows a variant of the synthetic reasoning task results in Subsection 1, where the symbols are instead sampled proportion to the token frequencies. Although the improvement still generally holds, the standard LM with character-based vocabulary becomes significantly better. We believe that this is because the model can get a significant advantage by guessing among the most common letter.

Dataset	Vocab	LookUp Acc		Permutation Acc	
		Standard	LI	Standard	LI
Pile	char	72.80	90.95	40.63	60.47
	32k	61.20	90.95	40.55	54.55
Wiki-40B	char	75.55	63.45	42.71	59.86
	32k	41.05	57.95	26.81	51.86
Github	char	66.00	86.75	36.62	70.77
	32k	59.25	78.45	37.46	65.04

Table 2: Synthetic Reasoning Tasks (adjusted for token frequencies)

G Language Models Regularized with Lexinvariance and BIG-bench Results

As described in the main paper, we implement a lexinvariance regularized Model in a way similar to embedding dropout. Note that one problem in implementing it naively by using random Gaussian embeddings and learned embedding in a mixture is that the two would become quickly distinguishable from each other during training since learned embeddings often have larger norms, allowing the model simply ignore the randomized tokens. So instead of using random Gaussian embedding matrices in place of a learned embedding matrix, we explored another approach for training a lexinvariant regularized LM: training a standard LM with learnable embedding matrix over sequences partially applied with a random token permutation $B_p(x_1;\ );\ldots;B_p(x_1;\ )$, where $B_p(x_i;\ ) = \ (x_i)$ with probability p and $B_p(x_i;\ ) = x_i$ with probability $1 \quad p$. Since each token can be remapped to any other token with equal chance, the produced model should ideally also be lexinvariant when $p = 1$, though with no strict guarantees. In practice, we found the models trained this way behave very similarly to models with random Gaussian embedding.

We evaluate our model over BIG-bench tasks where the language model performance scales well, and we prioritize evaluating generative tasks over multiple-choice tasks. Tasks we evaluated on:

gre reading comprehension.mul, linguistics puzzles.gen, linguistics puzzles.gen, rhyming.gen, tellmewhy.gen, simple arithmetic multiple targets json.gen, simple arithmetic json subtasks.gen, disfl qa.gen, arithmetic.gen, bridging anaphora resolution barqa.gen, matrixshapes.gen, sufficient information.gen, logical args.mul, novel concepts.mul, code line description.mul, unnatural in context learning.gen, unit interpretation.mul, english proverbs.mul, general knowledge.mul, geometric shapes.gen, human organs senses.mul, contextual parametric knowledge conflicts.gen, crass ai.mul,

auto categorization.gen, penguins in a table.gen, hindu knowledge.mul, english russian proverbs.mul, modified arithmetic.gen, cryobiology spanish.mul, evaluating information essentiality.mul, intent recognition.mul, understanding fables.mul, figure of speech detection.mul, empirical judgments.mul, simple ethical questions.mul, swahili english proverbs.mul, language identification.mul, phrase relatedness.mul, nonsense words grammar.mul, undo permutation.mul, object counting.gen, identify odd metaphor.mul, elementary math qa.mul, social iqa.mul, parsinlu qa.mul, metaphor understanding.mul, timedial.mul, causal judgment.mul, list functions.gen, implicatures.mul, date understanding.mul, codenames.gen, fact checker.mul, physics.mul, abstract narrative understanding.mul, emojis emotion prediction.mul, metaphor boolean.mul, strategyqa.gen, ascii word recognition.gen, auto debugging.gen, cause and effect.mul, conlang translation.gen, cryptonite.gen, cs algorithms.mul, dyck languages.mul, gender inclusive sentences german.gen, hindi question answering.gen, international phonetic alphabet transliterate.gen, irony identification.mul, logical fallacy detection.mul, movie dialog same or different.mul, operators.gen, paragraph segmentation.gen, parsinlu reading comprehension.gen, repeat copy logic.gen, rephrase.gen, simple arithmetic json.gen, simple arithmetic multiple targets json.gen, sports understanding.mul, word unscrambling.gen, hyperbaton.mul, linguistic mappings.gen, anachronisms.mul, indic cause and effect.mul, question selection.mul, hinglish toxicity.mul, snarks.mul, vitaminc fact verification.mul, international phonetic alphabet nli.mul, logic grid puzzle.mul, natural instructions.gen, entailed polarity.mul, list functions.gen, conceptual combinations.mul, goal step wikihow.mul, logical deduction.mul, conlang translation.gen, strange stories.mul, odd one out.mul, mult data wrangling.gen, temporal sequences.mul, analytic entailment.mul, disambiguation qa.mul, sentence ambiguity.mul, swedish to german proverbs.mul, logical sequence.mul, chess state tracking.gen, reasoning about colored objects.mul, implicit relations.mul, riddle sense.mul, physical intuition.mul, simple arithmetic json multiple choice.mul, geometric shapes.gen, gem.gen, simp turing concept.gen, common morpheme.mul, qa wikidata.gen, international phonetic alphabet transliterate.gen, similarities abstraction.gen, rephrase.gen, emoji movie.gen, qa wikidata.gen, word sorting.gen, emoji movie.gen, qa wikidata.gen, periodic elements.gen, hindi question answering.gen

Bellow, we plot the net percentage of tasks improved/deproved in each of the BIG-bench categories, out of the tasks that are changed by at least a threshold amount.

H Compute

We use one TPU v3-8 for all our pretraining runs. It takes approximately 23 hours for each pretraining run.

I Broader Impacts

Our work primarily provides a scientific exploration and understanding of the properties of lexinvariant language models. More broadly, these properties could potentially help improve the robustness, generalizability, and reasoning ability of LMs in the future works. In general we don't foresee more specific negative societal impacts from this work other than general misuse of language models.