Lexinvariant Language Models

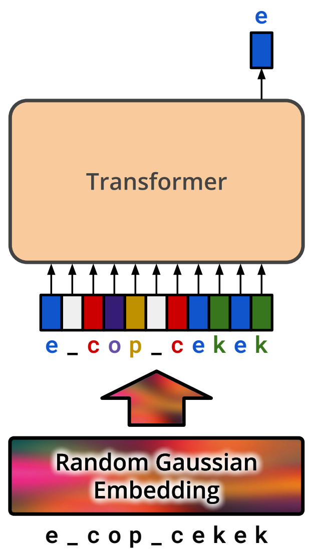
Anonymous Authors[1]

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 a radical question: could we build a performant language model without any fixed token embeddings? Such a language model would have to rely entirely on the structure of co-occurence and repetition of tokens in the context rather than the *a priori* identity of any token. To answer this, we introduce *lexinvariant* language models that do not have fixed token embeddings and are therefore invariant to lexical symbols. 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. We show that such a language model can still attain comparable perplexity to 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 demonstrate improvements on BIG-bench tasks.

`p("a big banana")`
`=`
`p("e cop cekeke")`
`=`
`p("o lan lomomo")`

(a) Lexinvariance



(b) Lexinvariant Language Model

Figure 1. Definition (a) and construction (b) of lexinvariant LM

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 (Schütze, 1993; Mikolov et al., 2013; 2010; Devlin et al., 2019; Brown et al., 2020). However, in real language use, 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 (Brown et al., 2020) and other large language models that demonstrate high lexical flexibility show strong performance on tasks involving in-context reasoning over new concepts and rules.

Motivated by the above, we ask a radical question in this paper: can we build a language model without *any* stable lexical mapping? We therefore formulate and study such lexinvariant language models. We define a lexinvariant language model as a language model that assigns the same probability to all lexical permutations of a sequence. Formally, we define a lexical permutation π to be a one-to-one-mapping of a set of lexical symbols onto itself. Then the

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

lexinvariant language model is defined as a language model over the symbol sequence $x_1, \ldots, x_n$ with the following property:

$$p(x_1, \ldots, x_n) = p(\pi(x_1), \ldots, \pi(x_n)) \ \forall \pi \qquad (1)$$

For example, a lexinvariant language model with letters as the set of all symbols should assign the same probability to the phrase `a big banana` as `e cop cekeke` and `o lan lomomo`, so that it is invariant to the lexical symbols used to form the phrase (Figure 1a). To achieve this, it needs to recognize that `e cop cekeke` is equivalent to `a big banana`, based only on the pattern of co-occurrences and repetition of letters in the phrase.

In practice, we train a lexinvariant language model by replacing standard embeddings in a decoder-only Transformer (Vaswani et al., 2017) 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 show more details in Sec. 2.1.

We show that the perplexity of this lexinvariant language model converges to the perplexity of a standard language model given a long enough context, both theoretically and empirically. Theoretically, we show that the conditional distribution learned by lexinvariant LM models converges to that of a standard language model as the context length becomes long enough to resolve possible ambiguities across lexical permutations. Empirically, we show that the gap of perplexity between the two shrinks across the context length. With a 150M decoder-only transformer Language Model and a small character-level vocabulary, the average perplexity gap shrinks from 9X to less than 1X the average perplexity of standard model after observing 512 tokens over a real corpus like the Pile. With a larger 32K vocabulary, the gap also shrinks, especially on more structured text like code on GitHub, but with much slower rate overall.

We then explore two properties of the lexinvariant language model: in-context deciphering and symbol manipulation. First, we show that given a ciphertext generated by applying a substitution cipher to English text, the lexinvariant language model 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 language model to predict the deciphered token corresponding to the last seen cipher token. We can then read out the inferred cipher key with each prefix of the sequence. We show that the accuracy of this inferred cipher key quickly improves as context length grows, reaching up to 99.6% average accuracy. And we also show several examples to visualize the uncertainties maintained by the lexinvariant language model when the cipher key is ambiguous, and that the semantic meaning of a rare symbol can be inferred efficiently 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 manipulations. We observe a significant 4X improvement over a standard language model. To generalize this performance gain to broader tasks that require understanding symbols that is not defined in-contex, we discuss several potential approaches to integrate the idea of lexinvariant LM to standard LM as a form of regularization, such that the LM assumes some form of partially stable symbol representations. We show that the resulting LM can improve upon a standard language model over some BIG-bench tasks.

2. Lexinvariant Language Model

We define a language model as a probability distribution $p(x_1, \ldots, x_n)$ over input token sequences $x_1, \ldots, x_n \in \mathcal{V}^n$, where $\mathcal{V}$ is some vocabulary that contains all possible tokens. The generated probability should capture the likelihood of the text represented by the input token sequence in a given corpus.

A language model is lexinvariant if for all permutations $\pi : \mathcal{V} \to \mathcal{V}$ and for all token sequences $x_1, \ldots, x_n \in \mathcal{V}^n$, $p(x_1, \ldots, x_n) = p(\pi(x_1), \ldots, \pi(x_n))$. For example, if $\mathcal{V} = \{\text{a}, \text{b}\}$, then the model should assign the same probability to aab and bba. One example p that satisfies this could simply be

$$p(x) = \begin{cases} 1/6 & x \in \{aab, bba\} \\ 1/6 & x \in \{aa, bb\} \\ 1/6 & x \in \{a, b\} \\ 0 & \text{otherwise} \end{cases}$$

2.1. Constructing a Lexinvariant Language Model

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 it parameterizes depends on the token embedding $E : \mathcal{V} \to \mathbb{R}^d$:

$$p(x_1, \ldots, x_n) = T(E(x_1), \ldots, E(x_n)) \qquad (2)$$

To make a neural LM lexinvariant, we can replace the standard stable token embedding E with a randomized E and take the expectation over E. Each token x in $\mathcal{V}$ has embedding $E(x) \sim \mathcal{N}(0, \mathcal{I}_d)$, and the language model becomes

$$p(x_1, \ldots, x_n) = \mathop{\mathbb{E}}_{E(x)} T(E(x_1), \ldots, E(x_n)) \qquad (3)$$

Since $E(x) \overset{d}{=} E(\pi(x))$, the right-hand side is the same when x_i are applied with any lexical permutation π, i.e.

$$\mathop{\mathbb{E}}_{E(x)} T(E(x_1), \ldots, E(x_n)) = \mathop{\mathbb{E}}_{E(\pi(x))} T(E(\pi(x_1)), \ldots, E(\pi(x_n))). \tag{4}$$

Now we can train this lexinvariant language model 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 $p(x_{n+1}|x_1, \ldots, x_n)$ instead of the joint distribution. Our definitions and analysis above still hold in general. The only difference 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 random embedddings of input tokens. 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 the input continuous representations as needed.

2.2. Convergence on Language Modeling Performance

Can this trained lexinvariant language model, which can only make next token predictions based on the structure of co-occurence and repetition of input tokens in a single context, model the language distribution well?

Although surprising at first, we show that it should indeed be able to model the true language distribution faithfully given long enough context that can resolve possible ambiguity across lexical permutations. As an intuitive example, suppose that $\mathcal{V} = \{a, b\}$ and the true language only contains two sequences $babbbb$ and $ababab$ (and their prefixes) with even probability. When given only the first three letters, the optimal lexinvariant model can only assign the same probability for the next letter: Due to the lexinvariant property, it assigns the same probability to $p(a|aba) = p(b|bab)$ as well as to $p(b|aba) = p(a|bab)$. Further, $p(a|aba) = p(b|aba)$ because the permutations to prefixes aba and bab are equally probable. In contrast, when considering the prefix $abab$, the fourth letter resolves the ambiguity in possible lexical permutations π. (Since $baba$ is not in the true language distribution, π cannot map a to b.) Therefore, the model can correctly predict that $p(a|abab) = 1$ and $p(b|abab) = 0$.

To show this formally, we first analyze the optimal lexinvariant language model produced by the training procedure

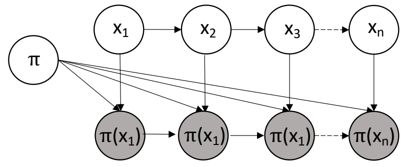


Figure 2. Probabilistic graphical model for training the lexinvariant language model.

described in the previous section. As shown in figure 2, we effectively train a standard language model to model a new sequence distribution $p'(\pi(x_1), \ldots, \pi(x_n))$, which is produced by sampling language sequence $x_1, \ldots, x_n$ from $p(x_1, \ldots, x_n)$ and sampling a symbol permutation π uniform randomly. More formally, we let $p'(x_1, \ldots, x_n)$ be the language modeling distribution produced by $\mathbb{E}_\pi \left(p(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n))\right)$.

Then, we have the following theorem:

Theorem 2.1. *For token sequences $x_1, \ldots, x_n$ generated by a language distribution p which we assume can be parameterized by a Hidden Markov Model with finite hidden states, we have that,*

$$\lim_{n\to\infty} \mathbb{E}_x \, ||p'(x_{n+1}|x_{1:n}) - p(x_{n+1}|x_{1:n})||_1 \to 0$$

We show its proof as well as a stronger version of this theorem in the Appendix. At a high level, this convergence happens because the ambiguity in lexical permutation is resolved as the sequence length goes towards infinity - we can bound the error by constructing a naive k-th order Markov model representing both p' and p by setting a sufficiently large window length. This bound leverages the result in Sharan et al. (2018), demonstrating the convergence of the expectation of language modeling loss to an arbitrary error ϵ with a naive Markov model as a function of the context window length l.

Therefore, an optimal lexinvariant model should converge to model the true language distribution eventually, given a long enough context that can resolve the possible ambiguity across lexical permutations. 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*!

Note that in practice, n often does not need to go to infinity for the $p(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n))$ to become very small for all π other than identity, as shown in the small example with two token vocab above. This horizon of convergence would obviously depend on the language distribution modeled. For math and code, where symbols typically define each other, the symbols should have relatively low ambiguity on what they represent in-context . The convergence horizon should also depend on the size of the tokenization vocabulary. For a vocabulary that is very large and contains tokens made of phrases or even sentences, there needs to be a long context length even to contain tokens that repeat, let alone to resolve ambiguity between them. On the other extreme, for a vocabulary of just bits (0 and 1), the lexinvariant language model should intuitively converge to the language distribution very quickly since swapping 0 and 1 does not change the information recorded at all. We explore this empirically in the experiment section.

2.3. In-context Bayesian Deciphering

We can see the 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:

$$\begin{aligned} &p'(x_{n+1}|x_1,\ldots,x_n) \\ &=\sum_{\pi}[\underbrace{p(\pi^{-1}(x_{n+1})|\pi^{-1}(x_1),\ldots,\pi^{-1}(x_n))}_{\text{language modeling}} \\ &\qquad\qquad \underbrace{\mathcal{P}(\pi|x_1,\ldots,x_n)}_{\text{inferring lexical permutation}}\ ] \end{aligned} \tag{5}$$

As shown above, p' 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 $\mathcal{P}(\pi|x_1,\ldots,x_n)$.

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

3. Experiments

3.1. Setup

Architecture. For all experiments, we use decoder-only Transformer architecture with T5 relative position bias (Raffel et al., 2022). We use models with 150M parameters, with 12 layers, 8 heads, head dimension 128, and MLP dimension 4096.

Training. We use the Adafactor optimizer (Shazeer & Stern, 2018), with a cosine decay learning rate schedule (Hoffmann et al., 2022) 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 (Bradbury et al., 2018).

Datasets. For datasets, we mainly use the Pile (Gao et al., 2020), a large open-source corpus that contains text col-

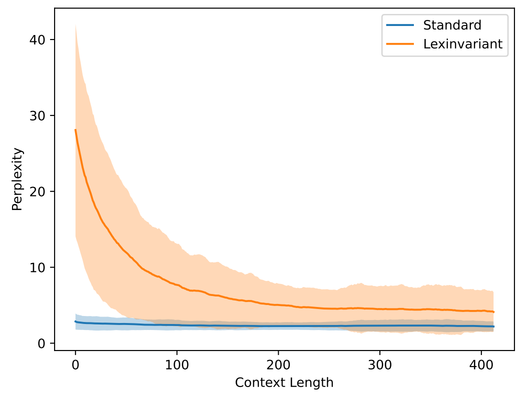


Figure 3. Token perplexity over the Pile with character-level vocab.

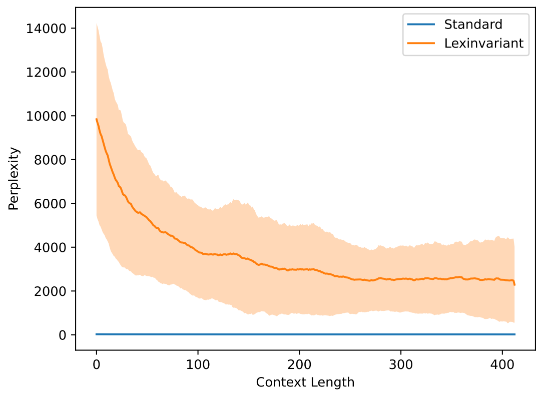


Figure 4. Token perplexity over the Pile with T5 default vocab.

lected 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 (Guo et al., 2020), 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.

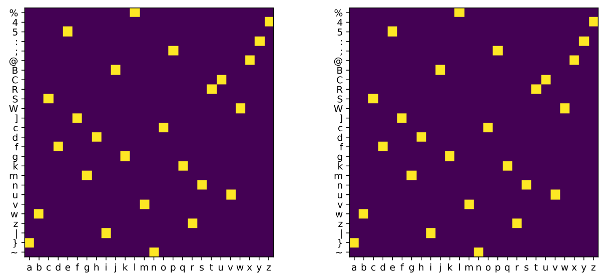
3.2. Convergence to Standard Language Models

We first show empirically that lexinvariant language models can mostly recover the next token prediction performance of regular language models after a long enough context. As already discussed in section 2.2, the lexinvariant language model will theoretically converge to a regular language model 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 language models with both character-level vocab (128 ascii characters) and T5 default vocab (32k tokens) over the three datasets. For each model, we measure the perplexity of each token w.r.t. context length, smoothed by moving average within each sequence, i.e. $P(x_i, \ldots, x_{i+k}|x_1, \ldots, x_i)^{\frac{1}{k}}$ for context length i. We set the moving average window $k = 100$. We plot results over 100 sequences. As shown in figure 3 and 4, the perplexity gap between lexinvariant language model and standard language model gradually shrinks as the prefix becomes longer and longer, albeit much more slowly with a larger vocab. This makes intuitive sense since a larger vocab has more possibilities of permutations and requires many more prefix tokens to disambiguate. For the 32K vocab, 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 vocab, the lexinvariant language model converges to nearly comparable perplexity to the standard language model. 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 this comparison across different datasets in Figure 9 in Appendix.

3.3. Recover Substitution Cipher Key

Here we show that lexinvariant language model is implicitly performing Bayesian in-context deciphering by testing its ability to recover cipher keys (e.g. Figure 5a) from character-level substitution ciphers, e.g. `uC; kvR5W 4mfzd @f| Svcgn fw;m uCRmu;;d ]%~} :fBn`. For the lexinvariant language model, this cipher text is perceived as the same 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`.



(a) Ground truth (b) Majority vote prediction

Figure 5. 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 `l`.

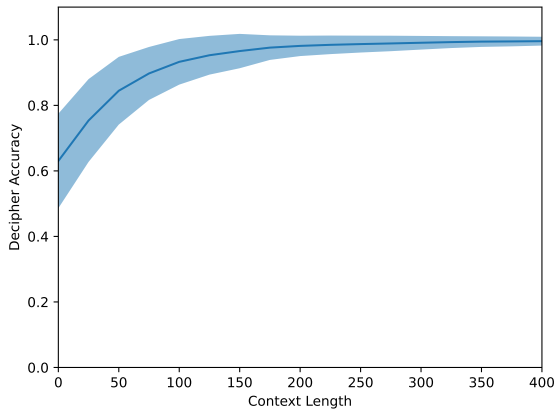


Figure 6. Cipher key prediction accuracy, averaged across 1000 input sequences. Context length denotes the start index of the window.

Because of this, we cannot directly read out the distribution of possible cipher keys $\mathcal{P}(\pi|x_1, \ldots, x_{n-1})$ implicitly inferred by the lexinvariant language model. To do this, we train a small two-layer MLP probe on top of a frozen trained lexinvariant language model. For each training sequence, we first embed the input sequence with a randomly sampled token embedding E as described in section 2.1 and obtain the hidden activation of the final layer generated by the frozen lexinvariant language model. 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 language model, 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 language model. We train the probe over the same corpus as the original lexinvariant language model for 10k steps. With this probe, we can directly visualize $\mathcal{P}(\pi^{-1}(x_n)|x_1, \ldots, x_n)$ inferred by the lexinvariant language model, which is effectively one row in the permutation matrix representing π.

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 5a. 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.

Concretely, we first input ciphertext through the frozen lexinvariant language model 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

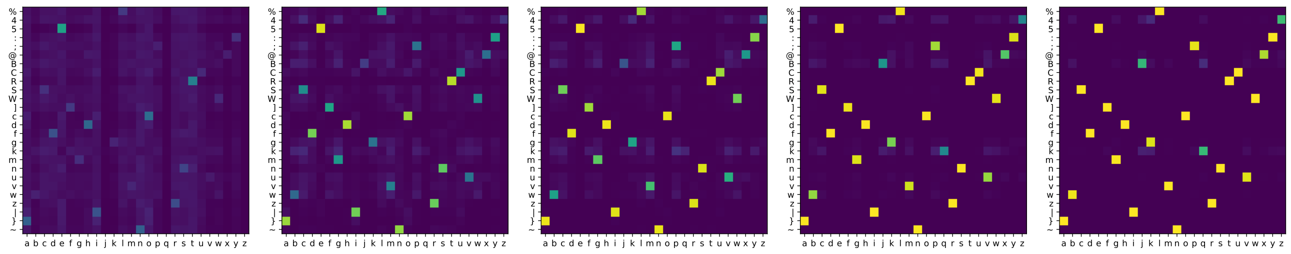


Figure 7. Predicted cipher key for windows of size 50, at indices 0, 50, 100, 200, and 400. Generated using temperature of $T = 1$.

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 6, 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 5b, we show the highly accurate predicted cipher key recovered from ciphertext produced using the example ground truth cipher key in Figure 5a.

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 $\mathcal{P}(\pi|x_1, \ldots, x_{n-1})$. 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 7, 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 language model 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.

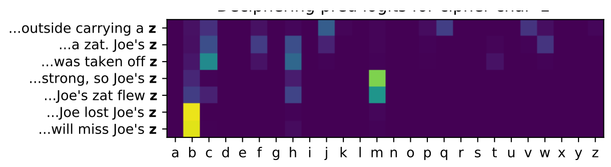
3.4.1. Uncertainty over Lexical Permutations

In Figure 8a, we input the following ciphered sequence to the frozen character-level lexinvariant language model 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 `a-z`.

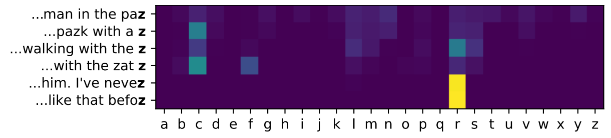
The lexinvariant model starts off assuming uniform probability for all possible lexical permutations π. After seeing more and more text, the lexinvariant model quickly realizes that `z` only has a few main plausible decipherings (`b`, `h`, `c`, `m`). Eventually, the lexinvariant model is able to narrow 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 8b 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 `r` after the appearance of "`right`". The disambiguation is reflected in the depicted probabilities.

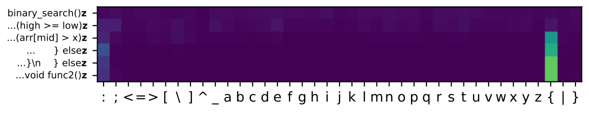
In Figure 8c and 8d, we show two deciphering examples over code. We consider two code examples in which it is initially ambiguous whether the character `z` deciphers to `:` or `{`. The ambiguity is eventually resolved by the use of Python-like or Java-like syntax.



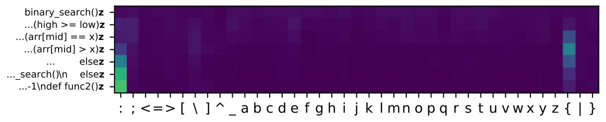
(a) True deciphering: “z” → “b”, $T = 1$.



(b) True deciphering: “z” → “r”, $T = 1$.



(c) True deciphering: “z” → “{”, $T = 2$.



(d) True deciphering: “z” → “:”, $T = 3$.

Figure 8. Probe predictions for deciphering “z” at each occurrence of “z” in context.

3.4.2. Semantic Deciphering

In addition to character-level deciphering, we show examples of semantic deciphering with the large vocababulary of 32k. Although the lexinvariant language model could not possibly figure out the true lexical permutation among 32k token using a small 512 context due to the large vocab size, it is possible to construct a simple context that repetitively use simple words so that these words are easier to decipher. Then the lexinvariant language model 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 flexibility is often correlated with in-context reasoning performance, as demonstrated by existing large language models. Thus, aside from the Bayesian deciphering, we study whether the lexinvariant model also learns other less obvious in-context reasoning capabilities through the challenging lexinvariant training.

Specifically, 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->` and Permutation, where the task is to permute an arbitrary subsequence of the given sequence following the given few demonstrations, e.g. `A 2 C->C A`⏎`4 1 D->`. In each of the tasks, the symbols are randomly sampled from the vocab so that we measure the pure reasoning ability independent from any knowledge of specific 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	TI	Standard	TI
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

3.6. Regularize Language Models with Lexinvariance

Although lexinvariant language model can gradually converge to the standard language model as shown in Sec. 2.2 and previous experiments, it would require the context to be extremely long so that all required words and knowledge are all defined in the context. Here, we discuss how to construct less extreme language models that maintain some properties of lexinvariant language models via regularization.

Instead of using random Gaussian embedding matrices in place of a learned embedding matrix all the time, we can use random embeddings for only some of the tokens in each sequence, while others use the learned embedding. This essentially means that the learned language model assumes that certain tokens have stable meanings based on information outside of the current context but not others, which can be seen as a form of regularization towards lexinvariant . There are several concrete ways to select which tokens to randomize the embedding for: 1) randomly selecting based on a Bernoulli distribution, which can essentially be seen as a form of dropout on token embeddings, such that the token meaning becomes more flexible; 2) selecting the low/high-frequency tokens, such that the more rare/common words are inferred in-context; 3) selecting the tokens that represent words belonging to open word classes, which commonly accept the addition of new words. Here, we mainly explore option 1. On 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. See full implementation details and results in the Appendix.

4. Related Work

4.1. Symbol Grounding

Beyond a modeling choice, the main question of our paper (that being whether a language model 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 (Harnad, 1990), with famous arguments like Searle's Chinese room. And this leads to an ongoing debate on whether LMs can learn meaning purely from large amounts of text, without grounding to any real-world objects (Bender et al., 2021). Conceptually, lexinvariant models are one step further away from physical grounding. We show that they can still potentially infer the meaning of symbols based on structures within the sequence context.

4.2. 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 (Xue et al., 2021). In the extreme, such a model would become closer and closer to lexinvariant language model, 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 language models with the lexinvariant property baked in and without requiring specific tokenizers.

4.3. Group invariances and Data augmentation

Our implementation of lexinvariant language models 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 (Jia & Liang, 2016; Akyürek et al., 2020) and augmentation of named entities in (Raiman & Miller, 2017), where certain parts of the sentence are swapped with other words while still maintaining the original grammatical structure. Different from these augmentations, our training for lexinvariant language models completely swaps out all parts of the input text.

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 language models that can assign probabilities to sequences (Nuhn & Ney, 2013). There have been several works focusing on solving substitution ciphers using LMs, including approaches from searching over the permutation space guided by LMs' scores (Hauer et al., 2014) to training a seq-to-seq model directly to perform deciphering as translation (Aldarrab & May, 2020). 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 (LLMs) acquire surprising in-context reasoning capabilities (Brown et al., 2020; Liang et al., 2022; Srivastava et al., 2022). 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 language model 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, based only on their co-occurrences and repetitions. 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 semi-lexinvariant language model and demonstrate its potential for solving more practical reasoning tasks efficiently.

References

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

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

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

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

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

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

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

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

Harnad, S. Symbol grounding problem. *Scholarpedia*, 2: 2373, 1990.

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

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

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

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

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech*, 2010.

Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

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

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jun 2022. ISSN 1532-4435.

Raiman, J. and Miller, J. Globally normalized reader. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

Roh, J., Oh, S.-H., and Lee, S.-Y. Unigram-normalized perplexity as a language model performance measure with different vocabulary sizes. *ArXiv*, abs/2011.13220, 2020.

Schütze, H. Part-of-speech induction from scratch. In *31st Annual Meeting of the Association for Computational Linguistics*, pp. 251–258, Columbus, Ohio, USA, June 1993. Association for Computational Linguistics. doi: 10.3115/981574.981608. URL https://aclanthology.org/P93-1034.

Sharan, V., Kakade, S., Liang, P., and Valiant, G. Prediction with a short memory. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1074–1087, 2018.

Shazeer, N. and Stern, M. Adafactor: Adaptive learning rates with sublinear memory cost. In Dy, J. G. and Krause, A. (eds.), *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*, pp. 4603–4611. PMLR, 2018. URL http://proceedings.mlr.press/v80/shazeer18a.html.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., Kluska, A., Lewkowycz, A., Agarwal, A., Power, A., Ray, A., Warstadt, A., Kocurek, A. W., Safaya, A., Tazarv, A., Xiang, A., Parrish, A., Nie, A., Hussain, A., Askell, A., Dsouza, A., Rahane, A. A., Iyer, A. S., Andreassen, A., Santilli, A., Stuhlmuller, A., Dai, A. M., La, A. D., Lampinen, A. K., 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, A., Roberts, B. R., Loe, B. S., Zoph, B., Bojanowski, B., Ozyurt, B., Hedayatnia, B., Neyshabur, B., Inden, B., Stein, B., Ekmekci, B., Lin, B. Y., Howald, B. S., Diao, C., Dour, C., Stinson, C., Argueta, C., Ram'irez, C. F., Singh, C., Rathkopf, C., Meng, C., Baral, C., Wu, C., Callison-Burch, C., Waites, C., Voigt, C., Manning, C. D., Potts, C., Ramirez, C. T., Rivera, C., Siro, C., Raffel, C., Ashcraft, C., Garbacea, C., Sileo, D., Garrette, D. H., 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., Mollo, D. C., Yang, D., Lee, D.-H., Shutova, E., Cubuk, E. D., Segal, E., Hagerman, E., Barnes, E., Donoway, E. P., Pavlick, E., Rodolà, E., Lam, E. F., Chu, E., Tang, E., Erdem, E., Chang, E., Chi, E. A., Dyer, E., Jerzak, E. J., Kim, E., Manyasi, E. E., Zheltonozhskii, E., Xia, F., Siar, F., Mart'inez-Plumed, F., Happ'e, F., Chollet, F., Rong, F., Mishra, G., Winata, G. I., de Melo, G., Kruszewski, G., Parascandolo, G., Mariani, G., Wang, G., Jaimovitch-L'opez, G., Betz, G., Gur-Ari, G., Galijasevic, H., Kim, H. S., Rashkin, H., Hajishirzi, H., Mehta, H., Bogar, H., Shevlin, H., Schütze, H., Yakura, H., Zhang, H., Wong, H., Ng, I. A.-S., Noble, I., Jumelet, J., Geissinger, J., Kernion, J., Hilton, J., Lee, J., Fisac, J. F., Simon, J. B., Koppel, J., Zheng, J., Zou, J., Koco'n, J., Thompson, J., Kaplan, J., Radom, J., Sohl-Dickstein, J. N., Phang, J., Wei, J., Yosinski, J., Novikova, J., Bosscher, J., Marsh, J., Kim, J., Taal, J., Engel, J., Alabi, J. O., Xu, J., Song, J., Tang, J., Waweru, J. W., Burden, J., Miller, J., Balis, J. U., Berant, J., Frohberg, J., Rozen, J., Hernández-Orallo, J., Boudeman, J., Jones, J., Tenenbaum, J. B., Rule, J. S., Chua, J., Kanclerz, K., Livescu, K., Krauth, K., Gopalakrishnan, K., Ignatyeva, K., Markert, K., Dhole, K. D., Gimpel, K., Omondi, K. O., Mathewson, K. W., Chiafullo, K., Shkaruta, K., Shridhar, K., McDonell, K., Richardson, K., Reynolds, L., Gao, L., Zhang, L., Dugan, L., Qin, L., Contreras-Ochando, L., Morency, L.-P., Moschella, L., Lam, L., Noble, L., Schmidt, L., He, L., Col'on, L. O., Metz, L., cSenel, L. K., Bosma, M., Sap, M., ter Hoeve, M., Andrea, M., Farooqi, M. S., 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., McElrath, M. A., Yee, M. A., Cohen, M., Gu, M., Ivanitskiy, M. I., 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, M., MukundVarma, T., Peng, N., Chi, N., Lee, N., Krakover, N. G.-A., Cameron, N., Roberts, N. S., Doiron, N., Nangia, N., Deckers, N., Muennighoff, N., Keskar, N. S., Iyer, N., Constant, N., Fiedel, N., Wen, N., Zhang, O., Agha, O., Elbaghdadi, O., Levy, O., Evans, O., Casares, P. A. M., Doshi, P., Fung, P., Liang, P. P., Vicol, P., Alipoormolabashi, P., Liao, P., Liang, P., Chang, P. W., Eckersley, P., Htut, P. M., Hwang, P.-B., Milkowski, P., Patil, P. S., Pezeshkpour, P., Oli, P., Mei, Q., LYU, Q., Chen, Q., Banjade, R., Rudolph, R. E., Gabriel, R., Habacker, R., Delgado, R. R., Millière, R., Garg, R., Barnes, R., Saurous, R. A., Arakawa, R., Raymaekers, R., Frank, R., Sikand, R., Novak, R., Sitelew, R., Bras, R. L., Liu, R., Jacobs, R., Zhang, R., Salakhutdinov, R., Chi, R., Lee, R., Stovall, R., Teehan, R., Yang, R., Singh, S. J., Mohammad, S. M., Anand, S., Dillavou, S., Shleifer, S., Wiseman, S., Gruetter, S., Bowman, S., Schoenholz, S. S., Han, S., Kwatra, S., Rous, S. A., Ghazarian, S., Ghosh, S., Casey, S., Bischoff, S., Gehrmann, S., Schuster, S., Sadeghi, S., Hamdan, S. S., Zhou, S., Srivastava, S., Shi, S., Singh, S., Asaadi, S., Gu, S. S., Pachchigar, S., Toshniwal, S., Upadhyay, S., Debnath, S., Shakeri, S., Thormeyer, S., Melzi, S., Reddy, S., Makini, S. P., hwan Lee, S., Torene, S. B., Hatwar, S., Dehaene, S., Divic, S., Ermon, S., Biderman, S. R., Lin, S. C., Prasad, S., Piantadosi, S. T., Shieber, S. M., Misherghi, S., Kiritchenko, S., Mishra, S., Linzen, T., Schuster, T., Li, T., Yu, T., Ali, T. A., Hashimoto, T., Wu, T.-L., Desbordes, T., Rothschild, T., Phan, T., Wang, T., Nkinyili, T., Schick, T., Kornev, T. N., Telleen-Lawton, T., Tunduny, T., Gerstenberg, T., Chang, T., Neeraj, T., Khot, T., Shultz, T. O., Shaham, U., Misra, V., Demberg, V., Nyamai, V., Raunak, V., Ramasesh, V. V., Prabhu, V. U., Padmakumar, V., Srikumar, V., Fedus, W., Saunders, W., Zhang, W., Vossen, W., Ren, X., Tong, X. F., Wu, X., Shen, X., Yaghoobzadeh, Y., Lakretz, Y., Song, Y., Bahri, Y., Choi, Y. J., Yang, Y., Hao, Y., Chen, Y., Belinkov, Y., Hou, Y., Hou, Y., Bai, Y., Seid, Z., Xinran, Z., Zhao, Z., Wang, Z. F., Wang, Z. J., Wang, Z., Wu, Z., Singh, S., and Shaham, U. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *ArXiv*, abs/2206.04615, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. 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

Let $p'(x_1, \ldots, x_n)$ be the language modeling distribution produced by $\mathbb{E}_\pi\left(p(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n))\right)$.

Theorem A.1. *For token sequences $x_1, \ldots, x_n$ generated by a language distribution p which we assume can be parameterized by a Hidden Markov Model with finite hidden states, and output alphabet of size d, we have that*

$$\lim_{n\to\infty} \mathbb{E}_x \, ||p'(x_{n+1}|x_{1:n}) - p(x_{n+1}|x_{1:n})||_1 \to 0$$

First, we restate Theorem 1 from (Sharan et al., 2018) for reference.

Lemma A.2. *(Sharan et al., 2018, Theorem 1) Suppose observations are generated by a Hidden Markov Model with at most s hidden states, and output alphabet of size d. Then for $\epsilon > 0$ there exists a window length $\ell = O(\frac{\log n}{\epsilon})$ and absolute constant c such that for any $T \geq d^{c\ell}$, if $t \in \{1, 2, \ldots, T\}$ is chosen uniformly at random, then the expected l_1 distance between the true distribution of x_t given the entire history (and knowledge of the HMM), and the distribution predicted by the naive "empirical" ℓ-th order Markov model based on $x_0, \ldots, x_{t-1}$ is bounded by $\sqrt{\epsilon}$.*

First, observe that this statement also applies to a permutation π of the alphabet applied to a token sequence (i.e. $\pi(x_1), \ldots, \pi(x_n)$), as any permutation is equivalent to a relabeling of the hidden states of the HMM. Also, observe that we can represent p' as an HMM with a central node with edges corresponding to the choice of π and then an HMM subgraph for each π, with probability corresponding to the probability of π. Specifically, for a naive ℓ-th order Markov model $\hat{p}$ and a token sequence $x_1, \ldots, x_n$ sampled from a language distribution p, we have by Lemma A.2 that

$$\mathbb{E}_{x,\pi} \, ||\hat{p}\left(\pi^{-1}(x_n)|\pi^{-1}(x_{n-1}), \ldots, \pi^{-1}(x_1)\right) - p'\left(\pi^{-1}(x_n)|\pi^{-1}(x_{n-1}), \ldots, \pi^{-1}(x_1)\right)||_1 \leq \sqrt{\epsilon}$$

.

Moreover, because we define $\hat{p}$ to be the naive ℓ-th order Markov model, we also have that

$$\mathbb{E}_{x,\pi} \, ||\hat{p}\left(\pi^{-1}(x_n)|\pi^{-1}(x_{n-1}), \ldots, \pi^{-1}(x_1)\right) - p\left(\pi^{-1}(x_n)|\pi^{-1}(x_{n-1}), \ldots, \pi^{-1}(x_1)\right)||_1 \leq \sqrt{\epsilon}$$

for the true language distribution p, by the same argument.

Therefore, under the same assumptions, since the l_1 distance defines a metric, we have by the triangle inequality that the expected l_1 distance between $p'\left(\pi(x_n)|\pi(x_{n-1}), \ldots, \pi(x_1)\right)$ and $p\left(\pi(x_n)|\pi(x_{n-1}), \ldots, \pi(x_1)\right)$ is at most twice the error $\sqrt{\epsilon}$ of the naive ℓ-th order Markov model. In particular, when π is the identity, we have our desired result.

Since we can always construct a naive ℓ-th order Markov model, the above argument shows that we can bound the l_1 distance between p' and p to any desired error ϵ by choosing a sufficiently large window length ℓ.

B. Convergence on other datasets

Figure 9 shows the perplexity of lexinvariant LMs across the three different datasets, where different vocabs are normalized by unigram probability following (Roh et al., 2020) to be comparable. 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.

C. 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
```

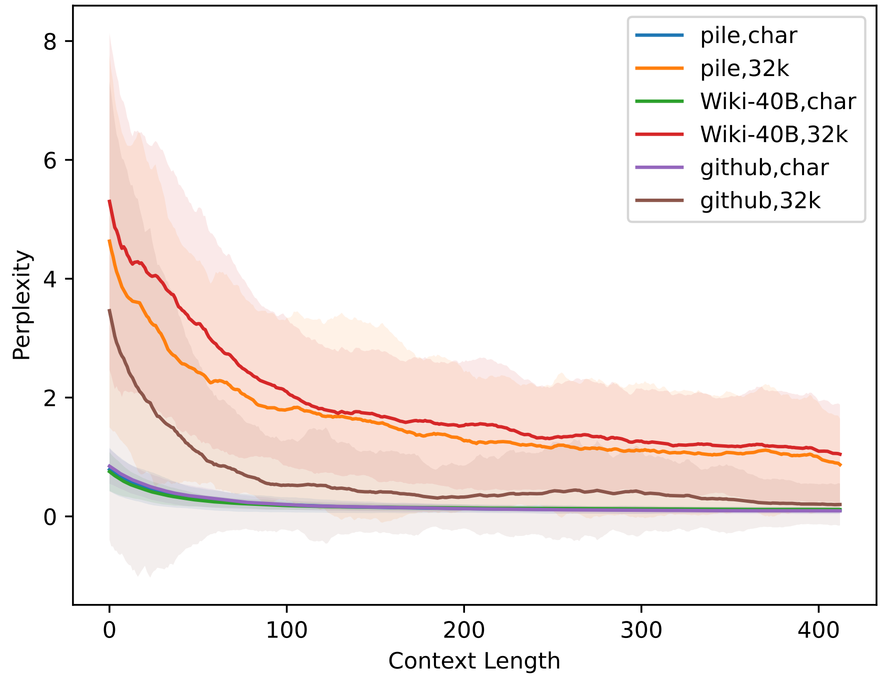


Figure 9. Smoothed and Unigram normalized Token Perplexity over the pile, Wiki-40B and Github, with character-level and T5 default vocab

```
        low = mid + 1;
        return binary_search();
      }
    } elsez
      return -1;
    }
}
void func2()z
```

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
```

D. 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
```

E. Synthetic Reasoning Task

Table 2 shows the results when the symbols are sampled based on the vocab frequencies. Although this makes a lot of sense for T5 standard vocab with 32K tokens, it unfairly boosted the performance of character-level vocab where the model can get a significant advantage from guessing among the most common letter.

Dataset	Vocab	LookUp Acc		Permutation Acc	
		Standard	TI	Standard	TI
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)

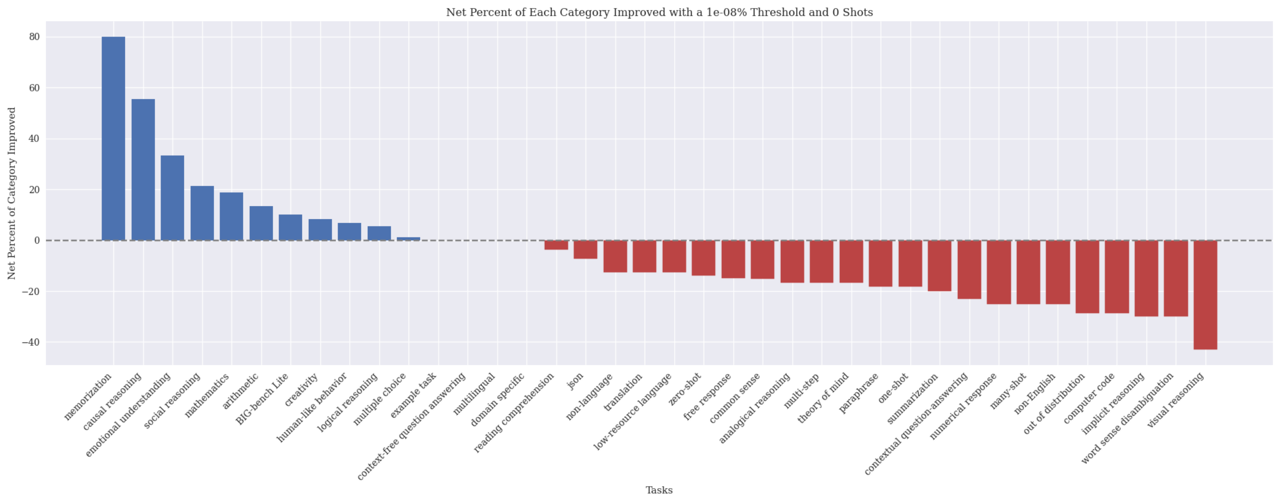
F. Language Models Regularized with Lexinvariance and BIG-bench Results

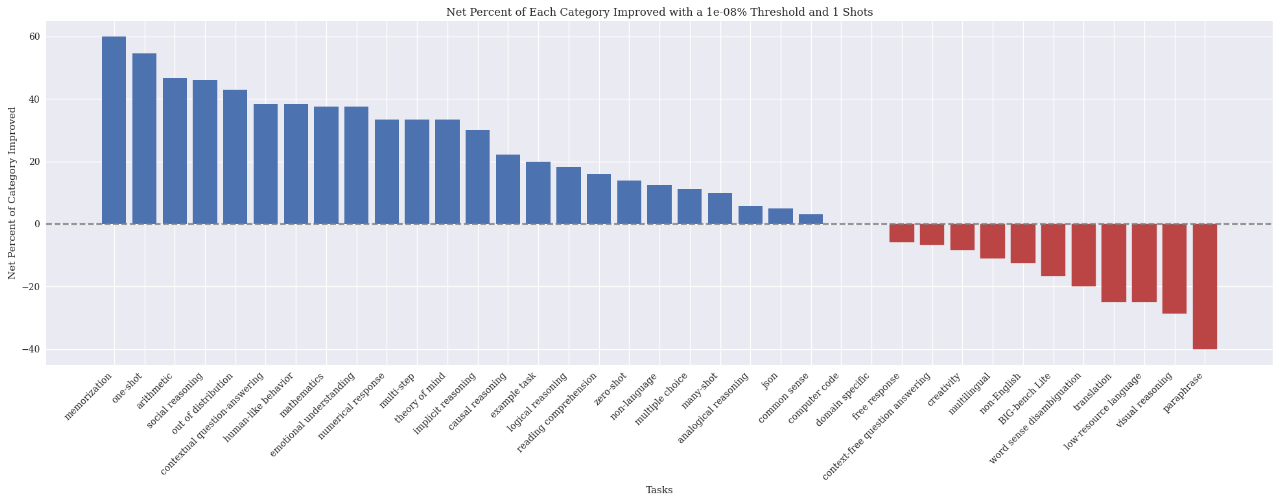
As described in the main paper, we implement a semi-lexinvariant Model in a way similar to embedding dropout. Note that one problem in implementing it naively by using random gaussian embedding and learned embedding in mix is that the two would become quickly distinguishable from each other during training since the learned embedding often has larger norms, making 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 semi-lexinvariant LM: training a standard LM with learnable embedding matrix over sequences partially applied with a random token permutation $B_p(x_1, \pi), \ldots, B_p(x_1, \pi)$, where $B_p(x_i, \pi) = \pi(x_i)$ with probability p and $B_p(x_i, \pi) = x_i$ with probability $1 - p$. 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 priorizes 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.


Net Percent of Each Category Improved with a 1e-08% Threshold and 0 Shots
Net Percent of Category Improved
80
60
40
20
0
−20
−40
memorization
causal reasoning
emotional understanding
social reasoning
mathematics
arithmetic
BIG-bench Lite
creativity
human-like behavior
logical reasoning
multiple choice
example task
context-free question answering
multilingual
domain specific
reading comprehension
json
non-language
translation
low-resource language
zero-shot
free response
common sense
analogical reasoning
multi-step
theory of mind
paraphrase
one-shot
summarization
contextual question-answering
numerical response
many-shot
non-English
out of distribution
computer code
implicit reasoning
word sense disambiguation
visual reasoning
Tasks


Net Percent of Each Category Improved with a 1e-08% Threshold and 1 Shots
Net Percent of Category Improved
60
40
20
0
−20
−40
memorization
one-shot
arithmetic
social reasoning
out of distribution
contextual question-answering
human-like behavior
mathematics
emotional understanding
numerical response
multi-step
theory of mind
implicit reasoning
causal reasoning
example task
logical reasoning
reading comprehension
zero-shot
non-language
multiple choice
many-shot
analogical reasoning
json
common sense
computer code
domain specific
free response
context-free question answering
creativity
multilingual
non-English
BIG-bench Lite
word sense disambiguation
translation
low-resource language
visual reasoning
paraphrase
Tasks

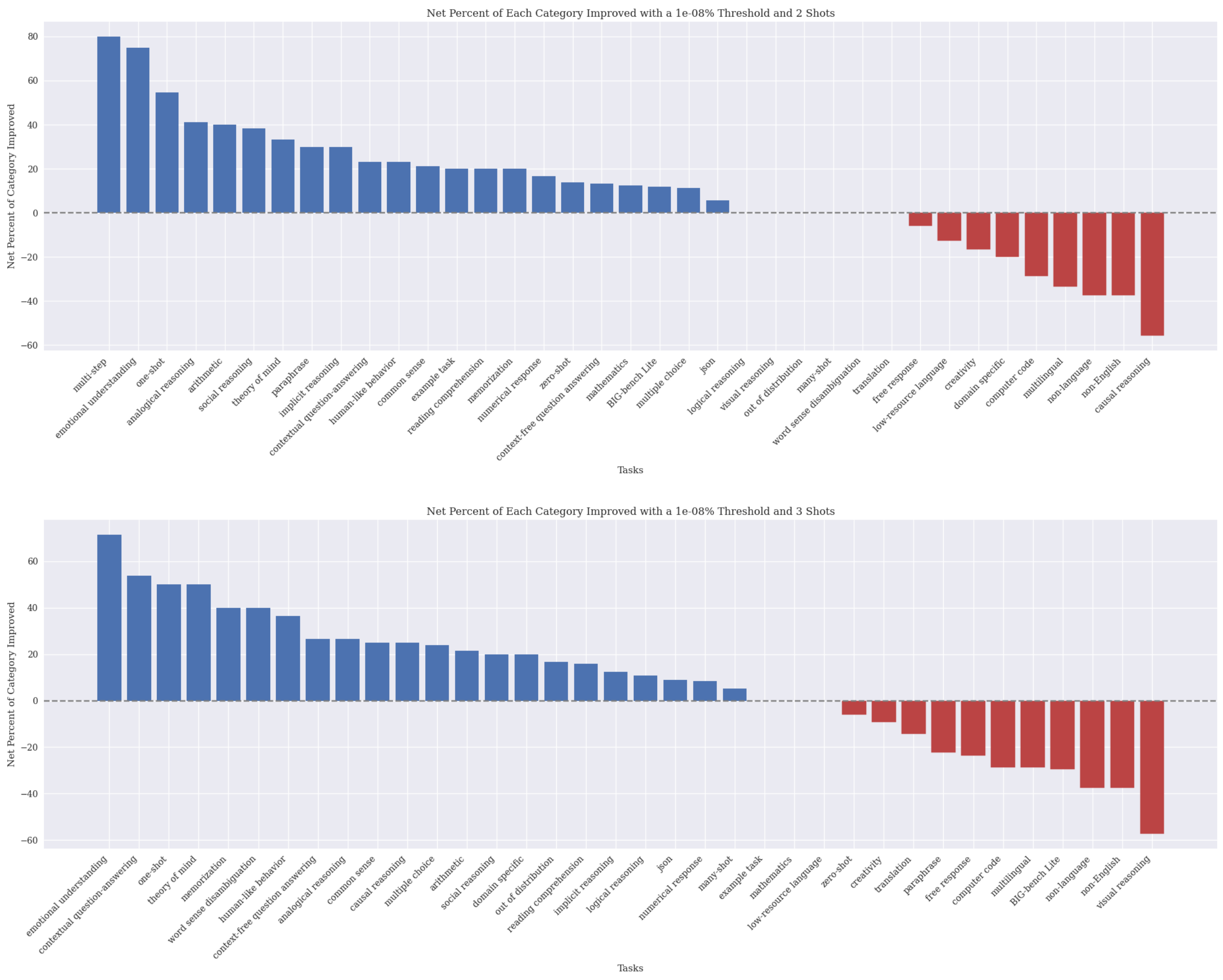


Figure 10. Caption