Interpretability Illusions in the Generalization of Simplified Models

Dan Friedman [1 *] **Andrew Lampinen** [2] **Lucas Dixon** [3] **Danqi Chen** [1 2] **Asma Ghandeharioun** [3]

Abstract

A common method to study deep learning systems is to use simplified model representations—for example, using singular value decomposition to visualize the model's hidden states in a lower dimensional space. This approach assumes that the results of these simplifications are faithful to the original model. Here, we illustrate an important caveat to this assumption: even if the simplified representations can accurately approximate the full model on the training set, they may fail to accurately capture the model's behavior out of distribution. We illustrate this by training Transformer models on controlled datasets with systematic generalization splits, including the Dyck balanced-parenthesis languages and a code completion task. We simplify these models using tools like dimensionality reduction and clustering, and then explicitly test how these simplified proxies match the behavior of the original model. We find consistent generalization gaps: cases in which the simplified proxies are more faithful to the original model on the in-distribution evaluations and less faithful on various tests of systematic generalization. This includes cases where the original model generalizes systematically but the simplified proxies fail, and cases where the simplified proxies generalize better. Together, our results raise questions about the extent to which mechanistic interpretations derived using tools like SVD can reliably predict what a model will do in novel situations.

1. Introduction

How can we understand deep learning models? Often, we begin by simplifying the model, or its representations, using tools like dimensionality reduction, clustering, and discretization. We then interpret the results of these simplifications—for example finding dimensions in the principal components that encode a task-relevant feature (e.g. Liu et al., 2022; Power et al., 2022; Zhong et al., 2023; Chughtai et al., 2023; Lieberum et al., 2023). In other words, we are essentially replacing the original model with a simplified proxy that uses a more limited—and thus easier to interpret—set of features. By analyzing these simplified proxies, we hope to understand at an abstract level how the system solves a task. Ideally, this understanding helps us predict model behavior in unfamiliar situations, and thereby anticipate failure cases or potentially unsafe behavior.

However, in order to arrive at understanding by simplifying a model, we have to assume that the result of the simplification is a relatively *faithful* proxy for the original model. For example, we need to assume that the principal components of the model representations, by capturing most of the variance, are thereby capturing the important details of the model's representations for its computations. In this work, we question whether this assumption is valid. First, some model simplifications, like PCA, are not computed solely from the model itself; they are calculated with respect to the model's representations for a particular collection of inputs, and therefore depend on the input data distribution. Second, even if a simplification does not explicitly depend on the training distribution, it might appear faithful on in-distribution evaluations, but fail to capture the model's behavior over other distributions. Thus, it is important to understand the extent to which simplified proxy models characterize the behavior of the underlying model beyond this restricted data distribution.

We therefore study how models, and their simplified proxies, generalize out-of-distribution. We focus on small-scale Transformer (Vaswani et al., 2017) language models trained on controlled datasets with systematic generalization splits.[1] We first consider models trained on the Dyck balanced-parenthesis languages. These languages have been studied in prior work on characterizing the computational expressivity of Transformers (e.g., Hewitt et al., 2020; Ebrahimi et al., 2020; Yao et al., 2021; Weiss et al., 2021; Murty et al., 2023; Wen et al., 2023), and admit a variety of sys-

* Work done while the author was a Student Researcher at Google Research. [1]Department of Computer Science, Princeton University [2]Google DeepMind [3]Google Research. Correspondence to: Dan Friedman <dfriedman@cs.princeton.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]That is, we focus on out-of-distribution settings where models can be expected to generalize to some extent.

tematic generalization splits, including generalization to unseen structures, different sequence lengths, and deeper hierarchical nesting depths. First, we simplify and analyze the model's representations, e.g. by visualizing their first few singular vectors. Next, for each simplification, we explicitly construct the corresponding simplified proxy models—for example, replacing the model's key and query representations with their projection onto the top-k singular vectors—and evaluate how the original models, and their simpler proxies, generalize to out-of-distribution test sets.

We find that the simplified proxy models are not as faithful to the original models out of distribution. While the proxies behave similarly to the original model on in-distribution evaluations, they reveal unexpected *generalization gaps* on out-of-distribution tests—the simplified model often *underestimates* the generalization performance of the original model, contrary to intuitions from classic generalization theory (Valiant, 1984; Bartlett & Mendelson, 2002), and recent explanations of grokking (Merrill et al., 2023). However, under certain data-independent simplifications the simpler model actually outperforms the original model; once again, this indicates a mismatch between the original model and its simplification. We elucidate these results by identifying some features of the model's representations that the simplified proxies are capturing and missing, and how these relate to human-written Transformer algorithms for this task (Yao et al., 2021). We show that different simplifications produce different kinds of divergences from the original model.

We finally test whether these findings extend to larger models trained on a more naturalistic setting: predicting the next character in a dataset of computer code. We test how models generalize to unseen programming languages and find generalization gaps in this setting as well, with the simplified models proving less faithful to the original model on out-of-domain examples. We also find that the results vary on different subsets of the code completion task, with larger generalization gaps on some types of predictions than others. These experiments offer some additional insight into how these gaps are related to different properties of sequence modeling tasks. Specifically, generalization gaps might be more pronounced on more "algorithmic" tasks, where the model must use a particular feature in a precise, context-dependent way. The effect is diminished in settings where various local features contribute to the model's prediction.

Our results raise a key question for understanding deep learning models: If we simplify a model in order to interpret it, will we still accurately capture model computations and behaviors out of distribution? We reflect on this issue; the related challenges in fields like neuroscience; and the relationship between complexity and generalization.

2. Setting

Transformer language models The Transformer (Vaswani et al., 2017) is a neural network architecture for processing sequence data. The input is a sequence of tokens $w_1, \ldots, w_N \in \mathcal{V}$ in a discrete vocabulary $\mathcal{V}$. At the input layer, the model maps the tokens to a sequence of d-dimensional embeddings $\boldsymbol{X}^{(0)} \in \mathbb{R}^{N \times d}$, which is the sum of a learned token embedding and a positional embedding. Each subsequent layer i consists of a multi-head attention layer (MHA) followed by a multilayer perceptron layer (MLP): $\boldsymbol{X}^{(i)} = \boldsymbol{X}^{(i-1)} + \mathrm{MHA}^{(i)}(\boldsymbol{X}^{(i-1)}) + \mathrm{MLP}^{(i)}(\boldsymbol{X}^{(i-1)} + \mathrm{MHA}^{(i)}(\boldsymbol{X}^{(i-1)}))$.[2] Following Elhage et al. (2021), multi-head attention (with H heads) can be written as

$$\mathrm{MHA}(\boldsymbol{X}) = \sum_{h=1}^{H} \mathrm{softmax}(\boldsymbol{X}\boldsymbol{W}_Q^h(\boldsymbol{X}\boldsymbol{W}_K^h)^\top)\boldsymbol{X}\boldsymbol{W}_V^h\boldsymbol{W}_O^h,$$

where $\boldsymbol{W}_Q^h, \boldsymbol{W}_K^h, \boldsymbol{W}_V^h \in \mathbb{R}^{d \times d_h}$ are referred to as the *query*, *key*, and *value* projections respectively, and $\boldsymbol{W}_O^h \in \mathbb{R}^{d_h \times d}$ projects the output value back to the model dimension. The MLP layer operates at each position independently; we use a two-layer feedforward network: $\mathrm{MLP}(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_1)\boldsymbol{W}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d_m}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_m \times d}$, and σ is the ReLU function. The output of the model is a sequence of token embeddings, $\boldsymbol{X}^{(L)} \in \mathbb{R}^{N \times d}$. We focus on autoregressive Transformer language models, which define a distribution over next words, given a prefix $w_1, \ldots, w_{i-1} \in \mathcal{V}$: $p(w_i \mid w_1, \ldots, w_{i-1}) \propto \exp(\theta_{w_i}^\top \boldsymbol{X}_{i-1}^{(L)})$, where $\theta_{w_i} \in \mathbb{R}^d$ is a vector of output weights for word w_i.

Dyck languages Dyck-k is the family of balanced parenthesis languages with up to k bracket types. Following the notation of Wen et al. (2023), the vocabulary of Dyck-k is the words $\{1, \ldots, 2k\}$, where, for any $t \in [k]$, the words $2t-1$ and $2t$ are the opening and closing brackets of type t, respectively. Given a sentence $w_1, \ldots, w_n$, the nesting depth at any position i is defined as the difference between the number of opening brackets in $w_{1:i}$ and the number of closing brackets in $w_{1:i}$. As in prior work (Yao et al., 2021; Murty et al., 2023; Wen et al., 2023), we focus on bounded-depth Dyck languages (Hewitt et al., 2020), denoted Dyck-(k, m), where m is the maximum nesting depth. We focus on Dyck for two main reasons. On one hand, the Dyck languages exhibit several fundamental properties of both natural and programming languages—namely, recursive, hierarchical structure, which gives rise to long-distance dependencies. For this reason, Dyck languages have been widely studied in prior work on the expressivity of Transformer language models (Hewitt et al., 2020; Yao et al., 2021), and in interpretability (Wen et al., 2023). On the

[2] The standard Transformer also includes a layer-normalization layer (Ba et al., 2016), which we omit here.

Table 1: Illustration of Dyck generalization splits. For simplicity, examples are drawn from Dyck-(3, 2). Three sample sentences for each set, their respective sentence structure, and nesting depth are shown below. O and C refer to open and closed brackets in the sentence structure.

Subset		Illustrative Samples			
Train	Random samples from Dyck-(3,2) with three different bracket types of `()` `[]` {}. All sentences have the maximum nesting depth of two.	Sentence	`()`{}	`[[]()]`	{}({}())
		Structure	OCOC	OOCOCC	OCOOCOCC
		Depth	1111	122221	11122221
Seen Struct	Random samples with bracket structures that appeared in the Train set, but with different bracket types.	Sentence	`[]()`	`[][]`	`()`{`[]()`}
		Structure	OCOC	OOCOCC	OCOOCOCC
		Depth	1111	122221	11122221
Unseen Struct	Random samples with bracket structures that have not appeared in the Train set, but have the same maximum nesting depth of two.	Sentence	`()[]`{}	{`[]`}({})	{`()`{}`()`}
		Structure	OCOCOC	OOCCOOCC	OOCOCOCC
		Depth	111111	12211221	12222221
Unseen Depth	Examples with maximum nesting depth strictly greater than two.	Sentence	`()`{`[`{}`]`}	`[[]([])]`	{}{}({{}})
		Structure	OCOOOCCC	OOCOOCCC	OCOCOOOCCC
		Depth	11123321	12223321	1111123321

other hand, these languages are simple enough to admit simple, human-interpretable algorithms (see Section B.1).

For our main analysis, we train models on Dyck-(20, 10), the language with 20 bracket types and a maximum depth of 10, following (Murty et al., 2023). To create generalization splits, we follow Murty et al. (2023) and start by sampling a training set with 200k training sentences—using the distribution described by Hewitt et al. (2020)—and then generate test sets with respect to this training set. Next, we recreate the structural generalization split described by Murty et al. (2023) by sampling sentences and discarding seen sentences and sentences with *unseen* bracket structures (**Seen struct**). The bracket structure of a sentence is defined as the sequence of opening and closing brackets (e.g., the structure of `([])[]` is OOCCOC). The above sampling procedure results in a shift in the distribution of sentence lengths, with the *Seen struct* set containing much shorter sentences than the training set.[3] Therefore, we create two equal-sized structural generalization splits, **Unseen struct (len $\leq$ 32)** and **Unseen struct (len $>$ 32)**, by sampling sentences, discarding sentences with seen structures, and partitioning by length. Finally, we create a **Unseen depth** generalization set by sampling sentences from Dyck-(20, 20) and only keeping those sentences with a maximum nesting depth of at least 10. All generalization sets have 20k sentences. The different generalization splits are illustrated in Table 1 and more details are provided in Appendix A.1. Following Murty et al. (2023), we evaluate models' accuracy at predicting closing brackets that are at least 10 positions away from the corresponding opening bracket, and score the prediction by the closing bracket to which the model assigns the highest likelihood.

Code completion We also experiment with larger models trained on a more practical task: predicting the next character in a dataset of computer code. Code completion has become a common use case for large language models, with applications in developer tools, personal programming assistants, and automated agents (e.g. Anil et al., 2023; OpenAI, 2023). This task is a natural transition point from the Dyck setting, requiring both "algorithmic" reasoning (including bracket matching) and more naturalistic language modeling (e.g. predicting names of new variables). We train character-level language models on the CodeSearchNet dataset (Husain et al., 2019), which is made up of functions in a variety of programming languages. As in the Dyck languages, we define the bracket nesting depth as difference between the number of opening and closing brackets at each position, treating three pairs of characters as brackets (`()`, {}, `[]`). We train models on Java functions with a maximum nesting depth of three and construct two kinds of generalization split: Java functions with deeper nesting depths (**Java, unseen depth**); and functions with a seen depth but written in an unseen language (**JavaScript**, **PHP**, **Go**). See Appendix B.6 for more details. We evaluate the

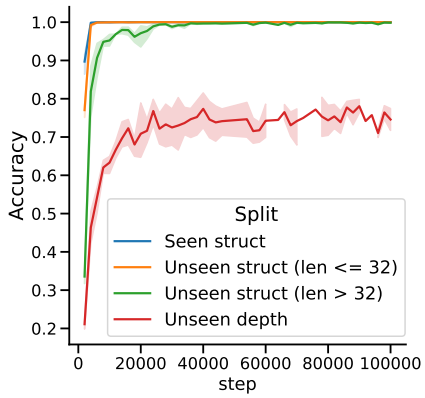


Figure 1: Accuracy at predicting closing brackets over the course of training, averaged over three random seeds.

[3]The longest sentence in our *Seen struct* set has a length of 30.

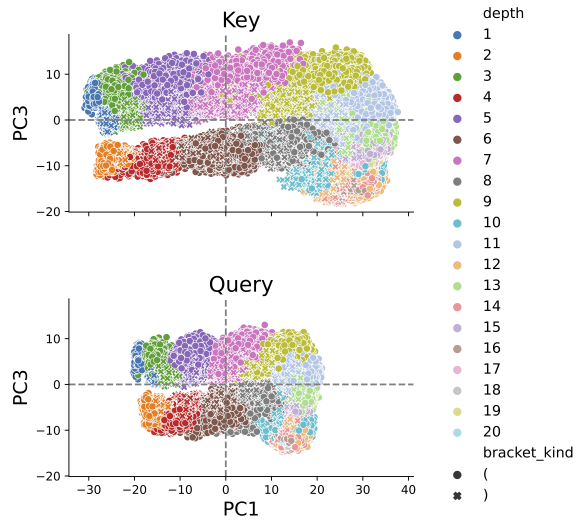


Figure 2: Second-layer attention embeddings for Dyck sequences, projected onto the first and third singular vectors and colored by bracket depth. The maximum depth during training is 10. At each position, the model can find the most recent unmatched bracket by finding the most recent bracket at the current nesting depth.

model's accuracy at predicting the next character, reporting the average over all characters in the evaluation set.

3. Approach

Scope of our analysis For Transformer LMs, mechanistic interpretations can be divided into two stages: *circuit identification* and *explaining circuit components*. The first stage involves identifying the subgraph of model components that are involved in some behavior. The second stage involves characterizing the computations of each component. Various automated circuit identification methods have been developed (e.g. Vig et al., 2020; Meng et al., 2022; Conmy et al., 2023), but relatively less work has focused on the second stage of interpretation, which we focus on here. In particular, we focus on characterizing the algorithmic role of individual attention heads.

Interpretation by model simplification Our main focus is on understanding individual attention heads, aiming to explain which features are encoded in the key and query representations and therefore determine the attention pattern. We evaluate two data-dependent methods of simplifying keys and queries, and a third data-agnostic method that simplifies the resulting attention pattern:

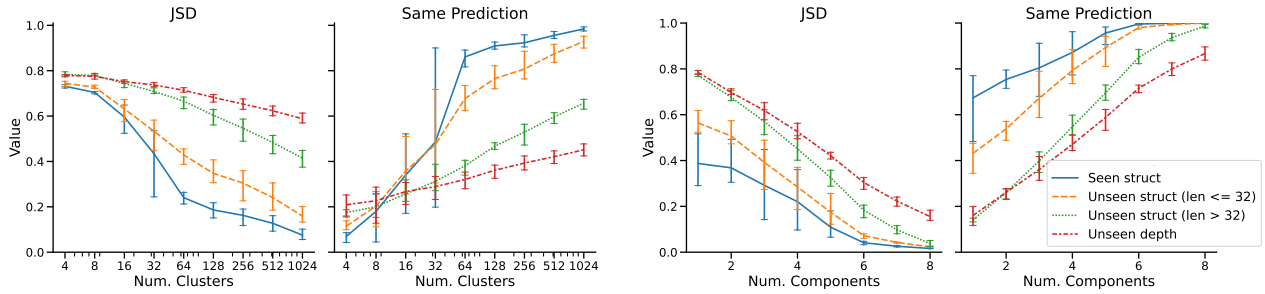
Simplifying key and query embeddings. Our first two approaches aim to characterize the attention mechanism by examining simpler representations of the key and query embeddings. To do this, we collect the embeddings for a sample of 1,000 training sequences. **Dimensionality reduction:** We calculate the singular value decomposition of the concatenation of key and query embeddings. For evaluation, we project all key and query embeddings onto the first k singular vectors before calculating the attention pattern.This approach is common in prior work in mechanistic interpretability (e.g. Lieberum et al., 2023). **Clustering:** We run k-means on the embeddings, clustering keys and queries separately. For evaluation, we replace each key and query with the closest cluster center prior to calculating the attention pattern. This approach has precedent in a long line of existing work on extracting discrete rules from RNNs (e.g. Omlin & Giles, 1996; Jacobsson, 2005; Weiss et al., 2018; Merrill & Tsilivis, 2022), and can allow us to characterize attention using discrete case analysis.

Simplifying the attention pattern. In the case of Dyck, prior work has observed that Transformers learn to use virtually hard attention in the final layer (Ebrahimi et al., 2020), assigning almost all attention to the most recent unmatched opening bracket. While our trained model uses standard softmax attention, this suggests a simplification where the soft attention is replaced with **one-hot attention** to highest-scoring key. This simplification has the advantage of being data-agnostic; it is purely a change to the model.

4. Case Study: Dyck Language Modeling

In this section, we train two-layer Transformer language models on Dyck languages. In Appendix B.2, we illustrate how we can attempt to reverse-engineer the algorithms these model learn by inspecting simplified model representations, using visualization methods that are common in prior work (e.g. Liu et al., 2022; Power et al., 2022; Zhong et al., 2023; Chughtai et al., 2023; Lieberum et al., 2023). In this section, we quantify how well these simplified proxy models predict the behavior of the underlying model. First, we plot approximation quality metrics for different model simplifications and generalization splits, finding a consistent generalization gap. Then we try to explain why this generalization gap occurs by analyzing the approximation errors. We include additional results in Appendix B.

Model and training details We train two-layer Transformer language models on the Dyck-(20, 10) training data described in the previous section. The model uses learned absolute positional embeddings. Each layer has one attention head, one MLP, and layer normalization, and the model has a hidden dimension of 32. Details about the model and training procedure are in Appendix A.2 and A.3. Fig. 1 plots the bracket-matching accuracy over the course of training, averaged over three runs. Consistent with Murty et al. (2023), we find that the models reach perfect accuracy on



(a) Clustering

(b) SVD

Figure 3: Approximation quality after applying two simplifications to the key and query embeddings, clustering (*left*) and SVD (*right*). *JSD* is the average Jensen-Shannon Divergence between the attention patterns of the original and simplified models, and *Same Prediction* measures whether the two models make the same prediction at the final layer. Plots show the mean and 95% CI after applying the simplification to models trained with three random seeds. For both methods, the approximation quality is better on the in-distribution evaluation set (*Seen struct*) and worse on examples with unseen structures or nesting depths.

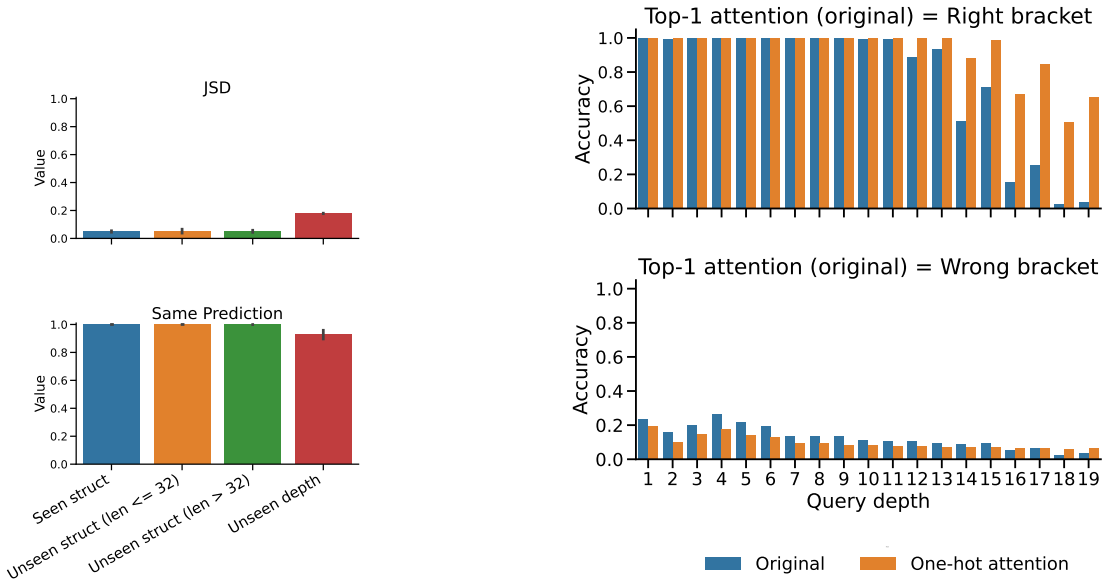
the in-domain held-out set early in training, and reach near-perfect accuracy on the structural generalization set later. On the depth generalization split, the models achieve approximately 75% accuracy. In Fig. 2 and Appendix B.3, we provide some qualitative analysis of the resulting models by examining low-dimensional representations of the attention embeddings. We find that the learned solutions resemble a human-written Transformer algorithm for Dyck (Yao et al., 2021), with the first attention layer using a broad attention pattern to calculate the bracket depth at each position, and the second layer using the depth to attend to the most recent unmatched bracket at each position. In the remainder of this section, we focus on the second attention layer, measuring the extent to which simplified representations approximate the original model on different generalization splits.

Generalization gaps We evaluate approximation quality using two metrics: the Jensen-Shannon Divergence (*JSD*), a measure of how much the attention pattern diverges from the attention pattern of the original model; and whether or not the two models make the *Same Prediction* at the final layer. Fig. 3 shows these metrics after simplifying the second-layer key and query representations. The simplified models correspond fairly well to the original model on the in-distribution evaluation set (*Seen struct*), but there are consistent performance gaps on the generalization splits. For example, Fig. 3b shows that we can reduce the key and query embeddings to as few as four dimensions and still achieve nearly 100% prediction similarity on the *Seen struct* evaluation set. However, whereas the original model generalizes almost perfectly to *Unseen structures*, the simplified model deviates considerably in these settings, suggesting that these simplification methods underestimate generalization. Fig. 4 shows the effect of replacing the attention pattern with one-hot attention. One-hot attention is a faithful approximation on all generalization splits except for the depth generalization split (Fig. 4a). In this setting, the one-hot attention model slightly out-performs the original model (Fig. 4b), in a sense over-estimating how well the model will generalize.

Comparing error patterns What do the simplified models miss? Figures 5a and 5b plot the errors made by the original model and a rank-8 SVD approximation, broken down by query depth and the maximum depth in the sequence. While both models have a similar overall error pattern, the rank-8 model somewhat underestimates generalization, performing poorly on certain out-of-domain cases where the original model successfully generalizes (i.e. depths 11 and 12). Figures 5c and 5d plot the errors made by the attention mechanism, showing the depth of the keys receiving the highest attention scores in cases where the final prediction of the original model is incorrect. In this case, both models have similar error patterns on shallower depths, attending to depths either two higher or two lower then the target depth. This error is in line with our visual analysis in the Appendix: in Appendix Fig. 9c, we can see that the attention embeddings encode the parity of the depth, with tokens at odd- and even-numbered depths having opposite signs in the third component. However, the error patterns diverge on depths greater than ten, suggesting that the lower-dimension model can explain why the original model makes mistakes in some in-domain cases, but not out-of-domain.

5. Case Study: Code Completion

Now we investigate whether our findings extend to larger models trained on a more practical task: predicting the next character in a dataset of computer code. This task is a natural transition point from the Dyck setting, requiring both "algorithmic" reasoning (including bracket matching) and more



(a) Approximation metrics for one-hot attention.

(b) Accuracy on the depth generalization split.

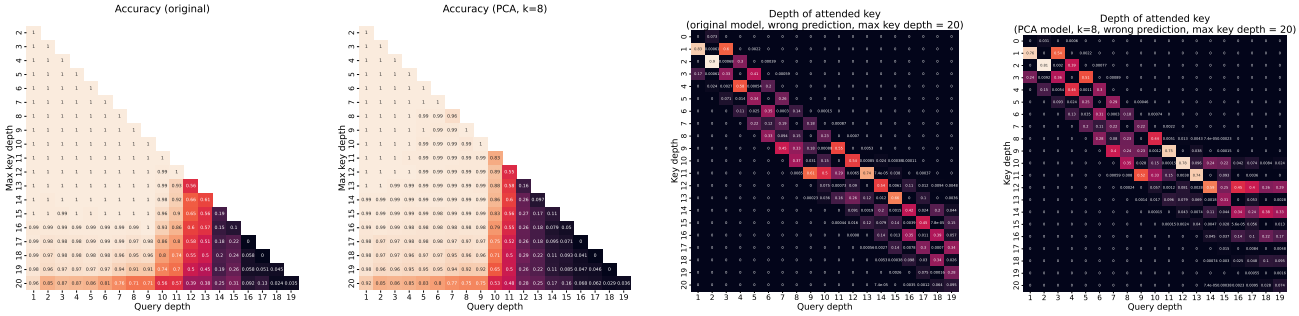
Figure 4: Approximation quality and accuracy after replacing the second-layer attention pattern with a one-hot attention pattern that assigns all attention to the highest scoring key, averaged over three models trained with different random seeds. One-hot attention is a faithful approximation on all generalization splits except for the depth generalization split (Fig. 4a). This difference illustrates that a simplification which is faithful in some out-of-distribution evaluations may fail in others. In depth generalization, the one-hot attention model slightly out-performs the original model (Fig. 4b)—particularly at higher depths and in cases where the original model attends to the correct location—thereby over-estimating how well the original model will generalize.

naturalistic language modeling. We train character-level language models on the CodeSearchNet dataset (Husain et al., 2019), training on Java functions with a maximum bracket nesting depth of three, and evaluating generalization to functions with deeper nesting depths (**Java, unseen depth**) and functions with seen depths but written in an unseen languages (**Javascript**, **PHP**, **Go**). We train Transformer LMs with four layers, four attention heads per-layer, and an attention embedding size of 64, and train models with three random initializations. Unlike Dyck, this task does not admit a deterministic solution; however, we find that there are enough regularities in the data to allow the models to achieve high accuracy at predicting the next token (Fig. 14). See Appendix B.6 for more training details.

Generalization gaps We measure the effect of simplifying each attention head independently: for each head, we reduce the dimension of the key and query embeddings using SVD and calculate the percentage of instances in which the original and simplified model make the same prediction. Figure 6a plots the average prediction similarity, filtering to cases where the original model is correct. The prediction similarity is consistently higher on Java examples and lower on unseen programming languages, suggesting that this task also gives rise to a generalization gap.[4] On the other hand, there is no discernible generalization gap on the Java depth generalization split; this could be because in human-written code, unlike in Dyck, bracket types are correlated with other contextual features, and so local model simplifications may have less of an effect on prediction similarity.

Comparing subtasks Which aspects of the code completion task give rise to bigger or smaller generalization gaps? In Figure 6b, we break down the results by the type of character the model is predicting: whether it is a **Whitespace** character; part of a reserved word in Java (**Keyword**); part of a new identifier (**New word**); part of an identifier that appeared earlier in the function (**Repeated word**); a **Close bracket**; or any **Other** character, including opening brackets, semicolons, and operators. We plot the results for each attention head in a single model after projecting the key and query embeddings to the first 16 SVD components and include results for other models and dimensions in Appendix B.6. The results vary depending on the prediction

[4]To verify the statistical significance of this result, we conducted a paired sample t-test, which indicated that the difference between ID and OOD approximation scores is statistically significant with $p \leq 0.005$. See Appendix B.6 for more details.



(a) Prediction errors (original). (b) Prediction errors (SVD). (c) Attention errors (original). (d) Attention errors (SVD).

Figure 5: Errors of the original model and a rank-8 SVD simplification on the depth generalization test set. Figures 5a and 5b plot the prediction accuracy, broken down by the depth of the query and the maximum depth among the keys. Figures 5c and 5d plot the depth of the token with the highest attention score, broken down by the true target depth, considering only incorrect predictions. The models have similar error patterns on shallower depths, attending to depths two higher or two lower than the target depth, but the simplified model diverges on depths greater than ten.

type, including both the overall approximation scores and the relative difference between approximation quality on different subsets. In particular, the generalization gap is larger on subtasks that can be seen as more "algorithmic", including predicting closing brackets and copying identifiers from earlier in the sequence. The gap is smaller when predicting whitespace characters and keywords, perhaps because these predictions rely more on local context.

Comparing attention heads In Figure 6b, we can observe that some prediction types are characterized by outlier attention heads: simplifying these attention heads leads to much lower approximation quality, and larger gaps in approximation quality between in-distribution and out-of-distribution samples. This phenomenon is most pronounced in the *Repeated word* category, where simplifying a single attention head reduces the prediction similarity score to 75% on in-domain samples and as low as 61% on samples from unseen languages. In the appendix (B.6), we find evidence that this head implements the "induction head" pattern, which has been found to play a role in the emergence of in-context learning in Transformer language models (Elhage et al., 2021; Olsson et al., 2022). This finding suggests that the low-dimensional approximation underestimates the extent to which the induction head mechanism will generalize to unseen distributions.

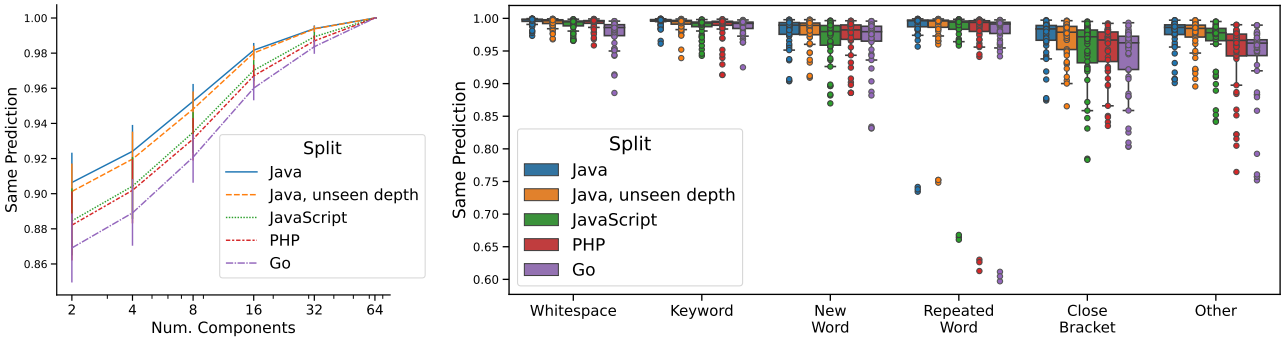
6. Discussion

In this section, we summarize our main experimental findings, discuss possible explanations for this phenomenon, and reflect on the broader implications.

Main findings Our experiments illustrated *generalization gaps*, cases where simplified proxy models are faithful to the original model on in-domain evaluations but fail to capture the model's behavior on tests of systematic generalization. On the Dyck languages (§4), simplifying attention embeddings (using clustering or PCA) resulted in models that generalized worse than the original model. Our analysis suggested that simplified embeddings capture coarse-grained features—for example, whether a bracket appears at an odd- or even-numbered nesting depth While this might be sufficient to explain model behavior in simpler in-domain settings, it fails to capture the finer-grained structure the model relies on to generalize. We observed similar generalization gaps on a code completion task (§2), with larger gaps on some subtasks than others. In particular, the largest gaps occur when the model must complete a word that appeared earlier in the sequence. We found evidence connecting this gap to an "induction head" mechanism; simplifying this attention head leads to a larger generalization gap on repeated words, indicating that the model relies on higher dimensional embeddings to implement this copying mechanism in out-of-distribution settings.

Why might such phenomena occur? One possible explanation for our results is that the gaps arise due to the data we use for simplifying the representations (to identify clusters or PCA components), and the gaps might disappear if we fit the approximations using OOD data as well. Our results suggest that such an approach might reduce generalization gaps in some cases. For example, from visualizing the Dyck embeddings in Figure 2, we might expecting the clustering simplification to be more faithful if the data used for clustering included examples with unseen depths.

On the other hand, even if the data used for simplification includes all relevant features, some features might be represented less strongly than others (in the sense of accounting for less variance between embeddings), despite being impor-



(a) Approximation quality.

(b) Approximation quality by prediction type with 16 components.

Figure 6: Prediction similarity on CodeSearchNet after reducing the dimension of the key and query embeddings using SVD, filtered to the subset of tokens that the original model predicts correctly. We apply dimensionality reduction to each attention head independently and aggregate the results over attention heads and over models trained with three random seeds. Each model has four layers, four attention heads per-layer, and a head dimension of 64. The prediction similarity between the original and simplified models is consistently higher on in-distribution examples (Java) relative to examples in unseen languages (Fig. 6a). Fig. 6b breaks down the results by prediction type, using 16 SVD components. Each point on the plot shows the prediction similarity after simplifying one attention head, and we plot all attention heads from all three random seeds. The gap between in-distribution and out-of-distribution approximation scores is greater on some types of predictions than others (e.g., *Repeated word*), perhaps because these predictions depend more on precise, context-dependent attention.

tant for generalization. For example, Lampinen et al. (2024) show that the strength with which a feature is represented can depend on a number of extraneous factors, including how common it is in the training data and how difficult it is to compute. In our case, simplified proxy models might capture only the most strongly-represented features. These might be sufficient to approximate the model on easier in-domain cases, but not on OOD settings that involve features that are rare or otherwise more weakly-represented.

Compression & simplification Hooker et al. (2019) observed that compressing vision models (via pruning) selectively reduces performance on a subset of examples; intriguingly, these examples also tend to be more challenging to classify, even for humans. Our results showing that simplified transformer models do worse on (more challenging) generalization splits may be an analogous phenomenon, but at the dataset rather than exemplar level (and for sequence data rather than images). From this perspective, our results may have implications for the common practice of compressing language models for more efficient serving, e.g. by quantization (e.g. Bhandare et al., 2019; Yao et al., 2023).

Related challenges in neuroscience Neuroscience also relies on stimuli to drive neural responses, and thus similarly risks interpretations that may not generalize to new data distributions. For example, historical research on retinal coding used simple bars or grids as visual stimuli. However, testing naturalistic stimuli produced many new findings (Karamanlis et al., 2022)—e.g., some retinal neurons respond only to an object moving differently than its background (Ölveczky et al., 2003), so their function could never be determined from simple stimuli. Thus, interpretations of retinal computation drawn from simpler stimuli did not fully capture its computations over all possible test distributions. Neuroscience and model interpretability face common challenges from interactions between different levels of analysis (Churchland & Sejnowski, 1988): we wish to understand a system at an abstract algorithmic level, but its actual behavior may depend on low-level details of its implementation.

The relationship between complexity and generalization Classical generalization theory suggests that simpler models generalize better (Valiant, 1984; Bartlett & Mendelson, 2002), unless datasets are massive; however, in practice overparameterized deep learning models generalize well (Nakkiran et al., 2021; Dar et al., 2021). Recent theory has explained this via *implicit regularization* effectively simplifying the models (Neyshabur, 2017; Arora et al., 2019), e.g. making them more compressible (Zhou et al., 2018). Our results reflect this complex relationship between model simplicity and generalization. We find that data-dependent approaches to simplifying the models' representations impair generalization. However, a *data-independent* simplification (hard attention) allows the simplified model to generalize better to high depths. This result reflects the match between the Transformer algorithm for Dyck and the inductive bias of hard attention, and therefore echoes some of the classical understanding about model complexity and the bias-variance tradeoff.

7. Related Work

Circuit analysis Research on reverse-engineering neural networks comes in different flavors, each focusing on varying levels of granularity. A growing body of literature aims to identify Transformer *circuits* (Olsson et al., 2022). Circuits refer to components and corresponding information flow patterns that implement specific functions, such as indirect object identification (Wang et al., 2023), variable binding (Davies et al., 2023), arithmetic operations (e.g., Stolfo et al., 2023; Hanna et al., 2023), or factual recall (e.g., Meng et al., 2022; Geva et al., 2023). More recently, there have been attempts to scale up this process by automating circuit discovery (Conmy et al., 2023; Davies et al., 2023) and establishing hypothesis testing pipelines (Chan et al., 2022; Goldowsky-Dill et al., 2023). In addition to the progress made in circuit discovery, prior research has highlighted some challenges. For example, contrary to earlier findings (Nanda et al., 2022), small models trained on prototypical tasks, such as modular addition, exhibit a variety of qualitatively different algorithms (Zhong et al., 2023; Pearce et al., 2023). Even in setups with strong evidence in favor of particular circuits, such as factual associations, modules that exhibit the highest causal effect in *storing* knowledge may not necessarily be the most effective choice for *editing* the same knowledge (Hase et al., 2023). Our results highlight an additional challenge: circuits identified with one dataset may behave differently out of distribution.

Analyzing attention heads Because attention is a key component of the Transformer architecture (Vaswani et al., 2017), it has been a prime focus for analysis. Various interesting patterns have been reported, such as attention heads that particularly attend to separators, syntax, position, or rare words (e.g., Clark et al., 2019; Vig, 2019; Guan et al., 2020). The extent to which attention can explain the model's predictions is a subject of debate. Some argue against using attention as an explanation for the model's behavior, as attention weights can be manipulated in a way that does not affect the model's predictions but can yield significantly different interpretations (e.g., Pruthi et al., 2020; Jain & Wallace, 2019). Others have proposed tests for scenarios where attention can serve as a valid explanation (Wiegreffe & Pinter, 2019). It is interesting to consider our results on one-hot attention simplification and depth generalization through the lens of this debate. Top-1 attention accuracy is highly predictive of model success in most cases; however, on deeper structures top-1 attention and accuracy dissociate. Thus, while top attention could be an explanation in most cases, it fails in others. However, our further analyses suggest that this is due to increased attention to other elements, even if the top-1 is correct—thus, attention patterns may still explain these errors, as long as we do not oversimplify attention before interpreting it.

Neuron-level interpretability A more granular approach to interpretability involves examining models at the neuron level to identify the concepts encoded by individual neurons. By finding examples that maximally activate specific "neurons"—a hidden dimension in a particular module like an MLP or in the Transformer's residual stream—one can deduce their functionality. These examples can be sourced from a dataset or generated automatically (Belinkov & Glass, 2019). It has been observed that a single neuron sometimes represents high-level concepts consistently and even responds to interventions accordingly. For example, (Bau et al., 2020) demonstrated that by activating or deactivating the neuron encoding the "tree" concept in an image generation model, one can respectively add or remove a tree from an image. Neurons can also work as n-gram detectors or encode position information (Voita et al., 2023). However, it is essential to note that such example-dependent methods could potentially lead to illusory conclusions (Bolukbasi et al., 2021).

Robustness of interpretability methods Our findings join a broader line of work in machine learning on the robustness of interpretability methods. For example, prior work in computer vision has found that explanation methods for computer vision can be both sensitive to perceptually indistinguishable perturbations of the data (Ghorbani et al., 2019), and insensitive to semantically meaningful changes (Adebayo et al., 2018), highlighting the importance of rigorous evaluation for interpretability methods.

8. Conclusion

In this work, we simplified Transformer language models using tools like dimensionality reduction, in order to investigate their computations. We then compared how the original models and their simplified proxies generalized out-of-distribution. We found consistent generalization gaps: cases in which the simplified proxy model is more faithful to original model on in-distribution evaluations and less faithful on various systematic distribution shifts. Overall, these results highlight the limitations of interpretability methods that depend upon simplifying a model, and the importance of evaluating model interpretations out of distribution.

Limitations This study focused on small-scale models trained on a limited range of tasks. Future work should study how these findings apply to larger-scale models trained on other families of tasks, such as large (natural) language models. Likewise, we only explored simplifying one component of a model at a time—such as a single attention head. However, model interpretations often involve larger circuits, and simplifying an entire circuit simultaneously might yield more dramatic distribution shifts; future work should explore this possibility.

Impact Statement

Methods for interpreting neural networks are important if we hope to deploy these systems safely and reliably. This work draws attention to a setting in which common approaches to interpretability can lead to misleading conclusions. These findings can caution against the pitfalls of these approaches and motivate the development of more faithful interpretability methods in the future.

Acknowledgments

We thank Adam Pearce, Adithya Bhaskar, and Mitchell Gordon for helpful discussions and feedback on writing and presentation of results.

References

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. Sanity checks for saliency maps. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. PaLM 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Arora, S., Cohen, N., Hu, W., and Luo, Y. Implicit regularization in deep matrix factorization. *Advances in Neural Information Processing Systems*, 32, 2019.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bartlett, P. L. and Mendelson, S. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Bau, D., Zhu, J.-Y., Strobelt, H., Lapedriza, A., Zhou, B., and Torralba, A. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1907375117.

Belinkov, Y. and Glass, J. Analysis methods in neural language processing: A survey. *Transactions of the Association of Computational Linguistics (TACL)*, 7:49–72, 2019.

Bhandare, A., Sripathi, V., Karkada, D., Menon, V., Choi, S., Datta, K., and Saletore, V. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*, 2019.

Bolukbasi, T., Pearce, A., Yuan, A., Coenen, A., Reif, E., Viégas, F., and Wattenberg, M. An interpretability illusion for BERT. *arXiv preprint arXiv:2104.07143*, 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.

Chan, L., Garriga-Alonso, A., Goldowsky-Dill, N., Greenblatt, R., Nitishinskaya, J., Radhakrishnan, A., Shlegeris, B., and Thomas, N. Causal scrubbing: A method for rigorously testing interpretability hypotheses. *Alignment Forum*, 2022.

Chughtai, B., Chan, L., and Nanda, N. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning (ICML)*, pp. 6243–6267. PMLR, 2023.

Churchland, P. S. and Sejnowski, T. J. Perspectives on cognitive neuroscience. *Science*, 242(4879):741–745, 1988.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, 2019.

Conmy, A., Mavor-Parker, A., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems (NeurIPS)*, 36: 16318–16352, 2023.

Dar, Y., Muthukumar, V., and Baraniuk, R. G. A farewell to the bias-variance tradeoff? An overview of the theory of overparameterized machine learning. *arXiv preprint arXiv:2109.02355*, 2021.

Davies, X., Nadeau, M., Prakash, N., Shaham, T. R., and Bau, D. Discovering variable binding circuitry with desiderata. *arXiv preprint arXiv:2307.03637*, 2023.

Ebrahimi, J., Gelda, D., and Zhang, W. How can self-attention networks recognize Dyck-n languages? In *Findings of Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4301–4306, 2020.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., et al. A mathematical framework for Transformer circuits. *Transformer Circuits Thread*, 2021.

Geva, M., Bastings, J., Filippova, K., and Globerson, A. Dissecting recall of factual associations in auto-regressive language models. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Ghorbani, A., Abid, A., and Zou, J. Interpretation of neural networks is fragile. In *Conference on Artificial Intelligence (AAAI)*, 2019.

Goldowsky-Dill, N., MacLeod, C., Sato, L., and Arora, A. Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*, 2023.

Guan, Y., Leng, J., Li, C., Chen, Q., and Guo, M. How far does BERT look at: Distance-based clustering and analysis of BERT's attention. In *International Conference on Computational Linguistics (COLING)*, pp. 3853–3860, 2020.

Hanna, M., Liu, O., and Variengien, A. How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Hase, P., Bansal, M., Kim, B., and Ghandeharioun, A. Does localization inform editing? Surprising differences in causality-based localization vs. knowledge editing in language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Hennigan, T., Cai, T., Norman, T., Martens, L., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Hewitt, J., Hahn, M., Ganguli, S., Liang, P., and Manning, C. D. RNNs can generate bounded hierarchical languages with optimal memory. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1978–2010, 2020.

Hooker, S., Courville, A., Clark, G., Dauphin, Y., and Frome, A. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248*, 2019.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Jacobsson, H. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17 (6):1223–1263, 2005.

Jain, S. and Wallace, B. C. Attention is not explanation. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 3543–3556, 2019.

Karamanlis, D., Schreyer, H. M., and Gollisch, T. Retinal encoding of natural scenes. *Annual Review of Vision Science*, 8:171–193, 2022.

Kazemnejad, A., Padhi, I., Natesan, K., Das, P., and Reddy, S. The impact of positional encoding on length generalization in Transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Lake, B. and Baroni, M. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning (ICML)*, pp. 2873–2882. PMLR, 2018.

Lampinen, A. K., Chan, S. C., and Hermann, K. Learned feature representations are biased by complexity, learning order, position, and more. *arXiv preprint arXiv:2405.05847*, 2024.

Lieberum, T., Rahtz, M., Kramár, J., Irving, G., Shah, R., and Mikulik, V. Does circuit analysis interpretability scale? Evidence from multiple choice capabilities in Chinchilla. *arXiv preprint arXiv:2307.09458*, 2023.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. In *International Conference on Learning Representations (ICLR)*, 2023.

Liu, Z., Kitouni, O., Nolte, N. S., Michaud, E., Tegmark, M., and Williams, M. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 34651–34663, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 17359–17372, 2022.

Merrill, W. and Tsilivis, N. Extracting finite automata from RNNs using state merging. *arXiv preprint arXiv:2201.12451*, 2022.

Merrill, W., Tsilivis, N., and Shukla, A. A tale of two circuits: Grokking as competition of sparse and dense subnetworks. *arXiv preprint arXiv:2303.11873*, 2023.

Murty, S., Sharma, P., Andreas, J., and Manning, C. Grokking of hierarchical structure in vanilla transformers. In *Association for Computational Linguistics (ACL)*, pp. 439–448, 2023.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2022.

Newman, B., Hewitt, J., Liang, P., and Manning, C. D. The EOS decision and length extrapolation. In *BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 276–291, 2020.

Neyshabur, B. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Ölveczky, B. P., Baccus, S. A., and Meister, M. Segregation of object and background motion in the retina. *Nature*, 423(6938):401–408, 2003.

Omlin, C. W. and Giles, C. L. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996.

OpenAI. GPT-4 technical report, 2023.

Patel, A., Bhattamishra, S., Blunsom, P., and Goyal, N. Revisiting the compositional generalization abilities of neural sequence models. In *Association for Computational Linguistics (ACL)*, pp. 424–434, 2022.

Pearce, A., Ghandeharioun, A., Hussein, N., Thain, N., Wattenberg, M., and Dixon, L. Do machine learning models memorize or generalize? *People+AI Research*, 2023.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Press, O. and Wolf, L. Using the output embedding to improve language models. In *European Chapter of the Association for Computational Linguistics (EACL)*, pp. 157–163, 2017.

Pruthi, D., Gupta, M., Dhingra, B., Neubig, G., and Lipton, Z. C. Learning to deceive with attention-based explanations. In *Association for Computational Linguistics (ACL)*, pp. 4782–4793, 2020.

Stolfo, A., Belinkov, Y., and Sachan, M. Understanding arithmetic reasoning in language models using causal mediation analysis. *arXiv preprint arXiv:2305.15054*, 2023.

Valiant, L. G. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems (NIPS)*, 30, 2017.

Vig, J. Visualizing attention in Transformer based language models. *arXiv preprint arXiv:1904.02679*, 2019.

Vig, J., Gehrmann, S., Belinkov, Y., Qian, S., Nevo, D., Sakenis, S., Huang, J., Singer, Y., and Shieber, S. Causal mediation analysis for interpreting neural NLP: The case of gender bias. *arXiv preprint arXiv:2004.12265*, 2020.

Voita, E., Ferrando, J., and Nalmpantis, C. Neurons in large language models: Dead, n-gram, positional. *arXiv preprint arXiv:2309.04827*, 2023.

Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: A circuit for indirect object identification in GPT-2 small. In *International Conference on Learning Representations (ICLR)*, 2023.

Weiss, G., Goldberg, Y., and Yahav, E. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning (ICML)*, pp. 5247–5256. PMLR, 2018.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like Transformers. In *International Conference on Machine Learning (ICML)*, pp. 11080–11090. PMLR, 2021.

Wen, K., Li, Y., Liu, B., and Risteski, A. Transformers are uninterpretable with myopic methods: A case study with bounded Dyck grammars. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023.

Wiegreffe, S. and Pinter, Y. Attention is not not explanation. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 11–20, 2019.

Yao, S., Peng, B., Papadimitriou, C., and Narasimhan, K. Self-attention networks can process bounded hierarchical languages. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pp. 3770–3785, 2021.

Yao, Z., Li, C., Wu, X., Youn, S., and He, Y. A comprehensive study on post-training quantization for large language models. *arXiv preprint arXiv:2303.08302*, 2023.

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zhou, W., Veitch, V., Austern, M., Adams, R. P., and Orbanz, P. Non-vacuous generalization bounds at the imagenet scale: A PAC-Bayesian compression approach. *arXiv preprint arXiv:1804.05862*, 2018.

A. Implementation Details:

A.1. Dyck Dataset Details

As described in Section 2, we sample sentences from Dyck-$(10, 20)$, the language of balanced brackets with 20 bracket types and a maximum nesting depth of 10. We use the sampling distribution described and implemented by Hewitt et al. (2020),[5] following existing work (Yao et al., 2021; Murty et al., 2023). We insert a special beginning-of-sequence token to the begin of each sequence, and append an end-of-sequence token to the end, and these tokens are included in all calculations of sentence length. Note that we discard sentences with lengths greater than 512. The training set contains 200k sentences and all the generalization sets contain 20k sentences. In all cases, we sample sentences, discarding sentences according to the rules described in Section 2, until the dataset has the desired size. See Table 1 for illustration of different generalization sets. Figure 7 plots the distribution of sentence lengths. For reference, we also include an **IID** split, which is created by sampling sentences from the same distribution used to construct the training set but rejecting any strings that appeared in the training set. It turns out that almost all sequences in this subset have unseen structures. (The number of bracket structures at length n is given by the nth Catalan number, so the chance of sampling the same bracket structure twice is very low.) For all of the experiments described in Section 4, results on this subset are nearly identical to results on the *Unseen structure (len $>$ 32)* subset.

A.2. Model Details

For our Dyck experiments, we use a two-layer Transformer, with each layer consisting of one attention head, one MLP, and one layer normalization layer. The model has a hidden dimension of 32, and the attention key and query embeddings have the same dimension. This dimension is chosen based on a preliminary hyperparameter search over dimensions in $\{2, 4, 8, 16, 32, 64\}$ because it was the smallest dimension to consistently achieve greater than 99% accuracy on the IID evaluation split. Each MLP has one hidden layer with a dimension of 128, followed by a ReLU activation. The input token embeddings are tied to the output token embeddings (Press & Wolf, 2017), and we use absolute, learned position embeddings. The model is implemented in JAX (Bradbury et al., 2018) and adapted from the Haiku Transformer (Hennigan et al., 2020).

[5] https://github.com/john-hewitt/dyckkm-learning

A.3. Training Details

We train the models to minimize the cross entropy loss:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{w_{1:n} \in \mathcal{D}} \frac{1}{n} \sum_{i=2}^{n} p(w_i \mid w_{1:i-1}),$$

where $\mathcal{D}$ is the training set, $p(w_i \mid w_{1:i-1})$ is defined according to Section 2, and w_1 is always a special beginning-of-sequence token. We train the model for 100,000 steps with a batch size of 128 and use the final model for further analysis. We use the AdamW optimizer (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-7, and a weight decay factor of 1e-4. We set the learning rate to follow a linear warmup for the first 10,000 steps followed by a square root decay, with a maximum learning rate of 5e-3. We do not use dropout.

B. Additional Results

B.1. Transformer Algorithms for Dyck

Our investigations will be guided by a human-written algorithm for modeling Dyck languages with Transformers (Yao et al., 2021), which we review here. The construction uses a two-layer autoregressive Transformer with positional encodings and one attention head per layer.

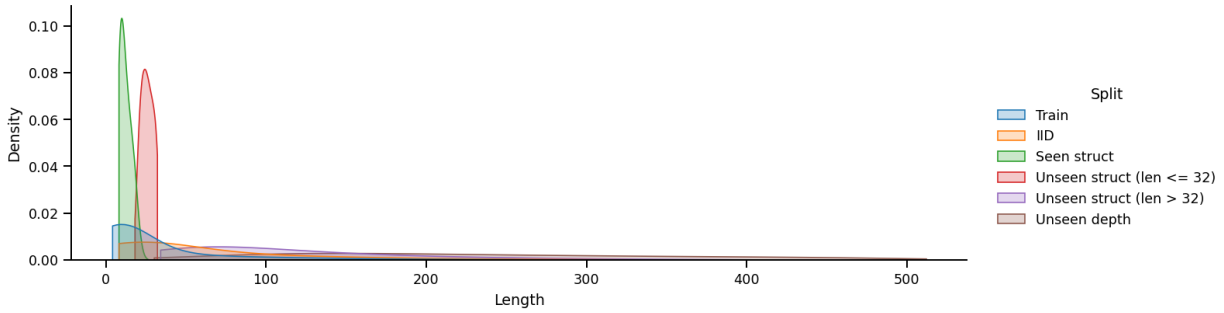
First attention layer: Calculate bracket depth. The first attention layer calculates the bracket depth at each position, defined as the number of opening brackets minus the number of closing brackets. One way to accomplish this is using an attention head that attends uniformly to all tokens and uses one-dimensional value embeddings, with opening brackets having a positive value and closing brackets having a negative value. At position t, the attention output will be $\mathrm{depth}(w_{1:t})/t$.

First MLP: Embed depths. The output of the first attention layer is a scalar value encoding depth. The first-layer MLP maps these values to orthogonal depth embeddings, which can be used as keys and queries in the next attention layer.

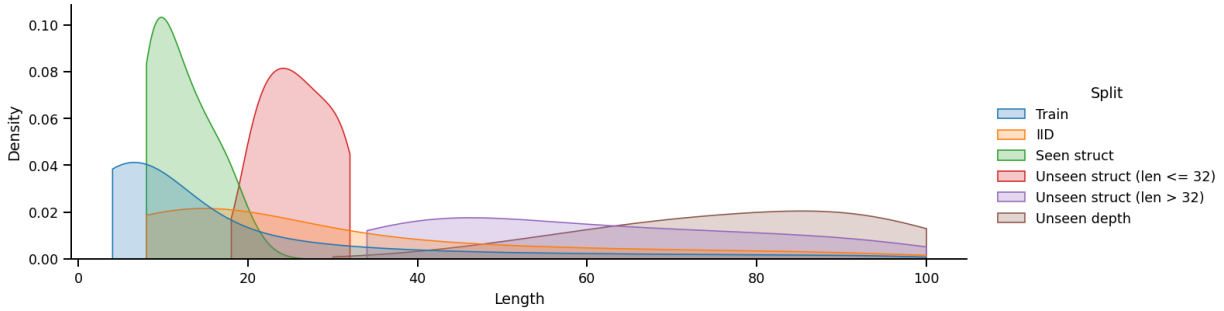
Second attention layer: Bracket matching. The second attention layer uses depth embeddings to find the most recent unmatched opening bracket. At position i, the most recent unmatched opening bracket is the bracket at position j, where $j \leq i$ is the highest value such that w_j is an opening bracket and $\mathrm{depth}(w_{1:j}) = \mathrm{depth}(w_{1:i})$.

B.2. Case Study: Analyzing a Dyck Language Model

In this section, we attempt to reverse-engineer the algorithm learned by a two-layer Dyck LM by analyzing simplified model representations, using visualization methods that are common in prior work (e.g. Liu et al., 2022; Power et al.,



(a) Length distributions (all lengths).



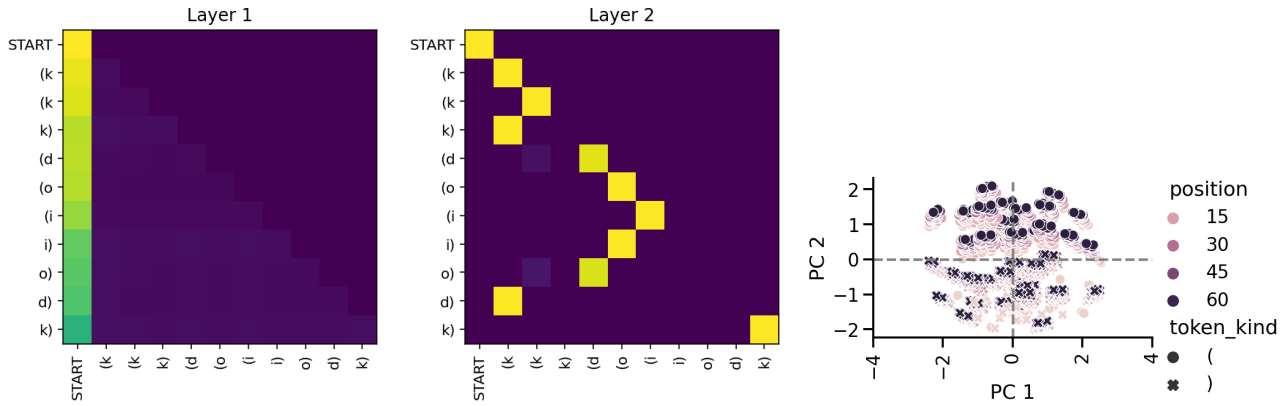
(b) Length distributions (lengths $\leq$ 100).

Figure 7: Distribution of sentence lengths in different Dyck structural generalization splits. Figure 7a shows the full distribution and Figure 7b shows the distribution filtered to sequences with length $\leq$ 100. These densities are estimated using the Seaborn library kernel density estimation method, with a bandwidth adjustment factor of three. (Note that all sequences of balanced parentheses have even lengths, which is smoothed over in the plots.) For reference, we also include an additional split created by sampling from the same distribution as the training data, but discarding sentences that appeared in the training set (**IID**); see Section A.1 for further discussion.

2022; Zhong et al., 2023; Chughtai et al., 2023; Lieberum et al., 2023).

First layer: Calculating bracket depth We start by examining the first attention layer. In Fig. 8, we plot an example attention pattern (Fig. 8a), along with the value embeddings plotted on the first two singular vectors (Fig. 8c). As in the human-written algorithm (§B.1), this attention head also appears to (1) attend broadly to all positions, and (2) associate opening and closing brackets with value embeddings with opposite sides. On the other hand, the attention pattern also deviates from the human construction in some respects. First, instead of using uniform attention, the model assigns more attention to the first position, possibly mirroring a construction from Liu et al. (2023). (In preliminary experiments, we found that the model learned a similar attention pattern even when we did not prepend a START token to the input sequences—that is, the first layer attention head attended uniformly to all positions but placed higher attention on the first token in the sequence.) Second, the value embeddings encode more information than is strictly needed to compute depth. Specifically, Figure 8c shows that the first components of the value embeddings encode position. Coloring this plot by bracket type reveals that the embeddings encode bracket type as well—each cluster of value embeddings corresponds to either opening or closing brackets for a single bracket type. In contrast, in the human-written Transformer algorithm, the value embeddings only encode whether the bracket is an opening or closing bracket and are invariant to position and bracket type This might suggest that the model is using some other algorithm, perhaps in addition to our reference algorithm. Despite these differences, these simplified representations of the model suggest that this model is indeed implementing some form of depth calculation.

Second layer: Bracket matching Now we move on to the second attention head. Behaviorally, this attention head appears to implement the bracket-matching function described in §B.1, and which has been observed in prior work (Ebrahimi et al., 2020); an example attention pattern is shown in Fig. 8b. Our goal in this section is to explain how this attention head implements this function in terms of the underlying key and query representations. Fig. 9 shows



(a) 1st layer attention pattern. (b) 2nd layer attention pattern (c) 1st layer value embeddings.

Figure 8: *Left:* An example attention pattern from the first-layer attention head. Each position attends broadly to the full input sequence, but with more attention concentrated on the beginning of sequence token. *Center:* An example attention pattern from the second-layer attention head. Each query assigns the most attention to the most recent unmatched opening bracket. *Right:* The first two singular vectors of the value embeddings. Each point represents the sum of a token embedding and a position embedding (for all token embeddings and positions up to 64).

PCA plots of the key and query embeddings from a sample of 1,000 training sequences. Again, these representations resemble the construction from Yao et al. (2021): the direction of the embeddings encodes depth (Fig. 9a), and the magnitude in each direction encodes the position, with later positions having higher magnitudes (Fig. 9b). On the other hand, the next two components (Fig. 9c) illustrate that the model also encodes the parity of the depth, with tokens at odd- and even-numbered depths having opposite signs in the third component. This reflects the fact that matching brackets are always separated by an even number of positions, so each query must attend to a position with the same parity.

Overall, this investigation illustrates how simple representations of the model can suggest an understanding of the algorithm implemented by the underlying model—in this case, that the model implements some version of the depth matching mechanism. (In Appendix B.3, we conduct a similar analysis using clustering and come to similar conclusions; see Fig. 10.)

B.3. Interpretation by Clustering

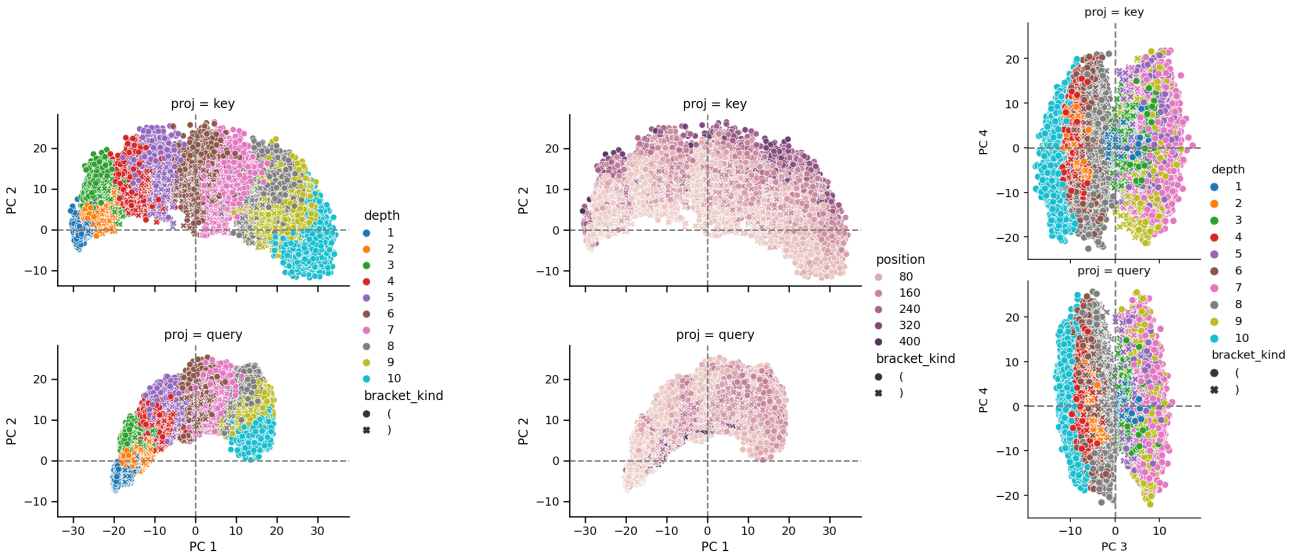
In Figure 10, we plot the distribution of depths associated with each cluster, after applying k-means separately to a sample of key and query embeddings, with k set to 16. Beneath each query cluster, we plot the depth distribution of the nearest key cluster, where distance is defined as the Euclidean distance between cluster centers. This figure illustrates how clustering allows us to interpret the model by means of a discrete case analysis: we can see that the queries are largely clustered by depth, and the nearest key cluster typically consists of keys with the same depth, suggesting that the model implements a (possibly imperfect) version of the depth-matching mechanism described in Section B.1.

B.4. Analysis: Structural Generalization

In Figure 11, we look at the approximation errors on the structural generalization test set (looking at the subset of examples with length ≤ 32). The figure plots the difference between the position with the highest attention score in the original model and simplified model. In both figures, the simplified version of the model generally attends to an earlier position than the true depth, and that differs from the target depth by an even number of positions. This suggests that these approximations "oversimplify" the model: they capture the coarse grained structure—namely, that positions always attend to positions with the same parity—but understimate the fidelity with which the model encodes depth.

B.5. Depth Generalization

Figure 12 plots key and query embeddings for out-of-distribution data points on the depth generalization split (Section 4). The success of the Transformer depends on the model's mechanism for representing nesting depth. To succeed on the depth generalization split, this mechanism must also extrapolate to unseen depths. Figure 5 indicates that the model does extrapolate to some extent, but the simplified models fail to fully capture this behavior. Figure 12 offers some hints about why this might be the case. These plots suggest that there is some systematic generalization (deeper depths are embedded where we might expect). However, we can also visually observe that the deeper depths



(a) Colored by depth.

(b) Colored by position.

(c) Singular vectors 3 & 4.

Figure 9: Key and query embeddings from the second-layer attention head, projected onto the first four singular vectors and colored by either bracket depth or position.

are less separated, perhaps indicating that the model uses additional directions for encoding deeper depths, but these directions are dropped when we fit SVD on data containing only lower depths. On the other hand, we did not have a precise prediction about where the deeper depths would be embedded, which is a limitation of our black box analysis: without reverse-engineering the lower layers, we cannot make strong predictions beyond the training data.

Are these findings consistent across training runs? In Figure 13, we train a Dyck model with a different random initialization and plot the prediction errors on the depth generalization, recreating Figure 5. The overall pattern is similar in this training run, with the simplified model diverging more from the original model on predictions at deeper query depths.

B.6. Additional Results: Computer Code

Dataset details We train character-level language models on functions from the CodeSearchNet dataset (Husain et al., 2019) with a maximum length of 512 characters. The vocabulary is defined as the ASCII printable characters; all other characters are replaced with a special *Unknown* token. The training data consists of Java functions with a maximum bracket nesting depth of three, where the bracket nesting depth is defined as the difference between the number of opening and closing brackets at each position and we treat three pairs of characters as brackets (`()`, {}, `[]`). The training examples are drawn from the original training split. We evaluate the models on: unseen Java functions with maximum depth of three (*Java*); unseen Java functions with maximum depth strictly greater than three (*Java, unseen depth*); and functions in unseen programming languages, with a maximum depth of three (*JavaScript*, *Go*, *PHP*). We draw the training examples from the original training split and the evaluation examples from the validation split. Table 2 reports the number of functions in each subset, and the average length in characters.

To break down the results by prediction type, we categorize each character as follows. First, we define a character as *Whitespace* if is a whitespace character (according to the Python `isspace` method) and a *Close bracket* if it is one of `)`, }, or `]`. Next, we split each function by whitespace and non-alphanumeric characters to obtain sequences of contiguous alphanumeric characters (*words*). We categorize a word as a *Keyword* if it is a Java reserved word,[6] or the word *true*, *false*, or *null*; a *Repeated word* if the word appeared earlier in the sequence; and a *New word* otherwise. Each character is assigned to the same category of the word it is a part of. Characters that are not part of a word—i.e., non-alphanumeric characters—are categorized as *Other*.

Model and training details We trained decoder-only Transformer language models with four layers and four attention heads per layer. The embedding size was 256, the attention embedding size was 64, and the MLP hidden layer was 512. We used a dropout rate of 0.1 and a batch size of

[6] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

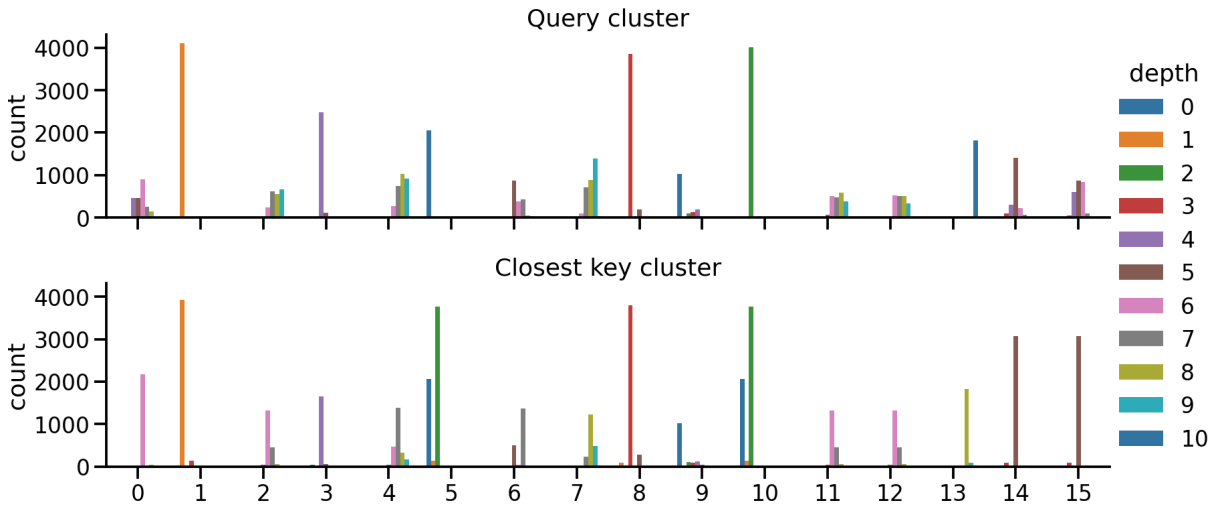


Figure 10: Interpreting the attention mechanism via clustering. The top row plots the depth distribution in each query cluster, and the bottom row plots the depth distribution in the key cluster closest to that query cluster (so some key clusters appear twice). As an interpretation of the model, this figure suggests that the key and query embeddings are reasonably well clustered by depth, and queries generally attend to clusters of the same depth, but the mechanism seems somewhat more effective for some depths (see clusters 1, 3, 8) then others, and in some cases the model might confuse a depth for another depth +/- 2 (clusters 5 and 10). The figure does not capture the mechanism for attending to the most recent token at a matching depth.

Table 2: The number of functions and the average length (in characters) of the training and evaluation subsets of CodeSearchNet. See Appendix B.6 for more details.

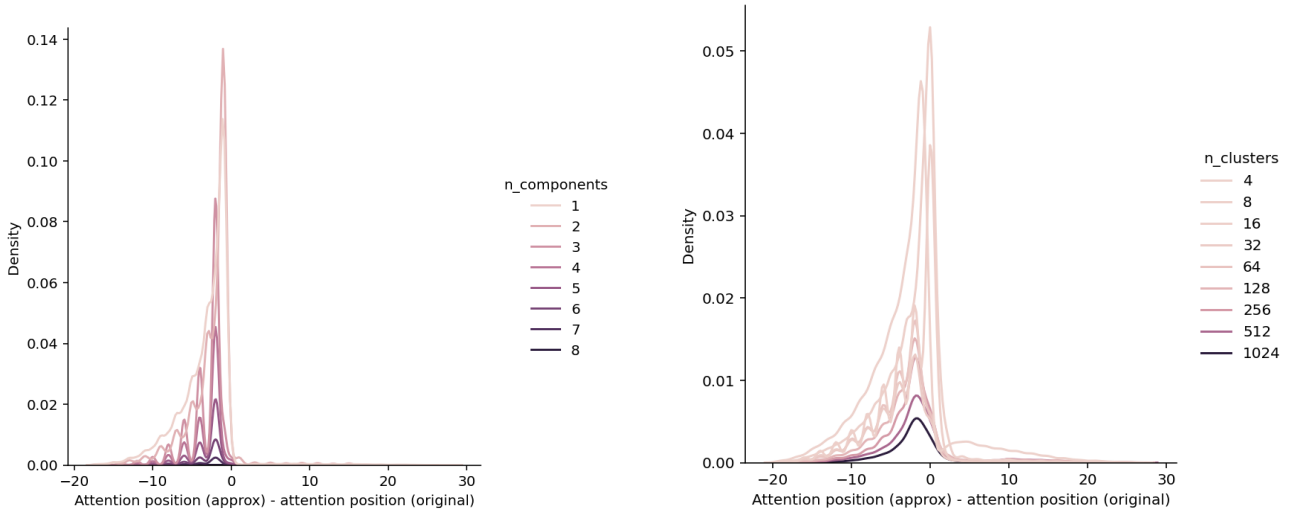
	Java (train)	**Java (eval)**	**Java, unseen depth**	**JavaScript**	**Go**	**PHP**
Num. examples	202,395	7,589	2,817	2,317	9,402	9,338
Avg. length	257	240	356	257	179	271

32 and trained for 100,000 steps. We set the learning rate to follow a linear warmup for the first 10,000 steps followed by a square root decay, with a maximum learning rate of 5e-3, and trained three models with different random initializations. Other model and training details are the same as described in Appendix A.3. Figure 14 plots the accuracy of the (original) models at predicting the next character at the end of training, broken down by the prediction type. The accuracy is relatively high, in part because many characters are whitespace, part of common keywords, or part of variable names that appeared earlier in the sequence.

Significance test Figure 6a depicts the generalization gap on the code completion task aggregated across attention heads as well as model initialization. To verify that the gap is statistically significant, we conducted a paired sample t-test[7] as follows. For every choice of model initialization, we measured the average approximation quality score (averaging over attention heads) on in-domain data (Java), and on out-of-domain data (aggregating the OOD splits). The null hypothesis for this test is that the expected difference between in-domain and OOD approximation scores is zero. We ran this analysis separately for each simplification level (number of SVD components). At each simplification level, this test finds that the expected difference between ID and OOD approximation scores is positive and the result is statistically significant with $p \leq 0.005$.

Additional approximation results We provide some additional results from the experiments described in Section 5. Figure 15 plots the prediction similarity on CodeSearchNet broken down by whether the underlying model's prediction is correct or incorrect. Prediction similarity is lower overall when the model is incorrect, although the gap between in-domain and out-of-domain approximation scores is generally smaller. Figure 16 breaks down the results by prediction type, showing the effect of simplifying each attention head, aggregated across models trained with three random initializations, each with four layers and four attention heads

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html



(a) SVD

(b) Clustering

Figure 11: These figures illustrate the kinds of errors in attention patterns that the approximations introduce on Unseen structure (length ≤ 32) evaluation split. They plot the distribution of distances between the position to which the original model assigns the most attention, and the position attended to by the simplified model. Both plots have a periodic structure, where the simplified model attends to a position that differs from the correct position by a multiple of two. This error pattern can be understood by noting that the parity of the nesting depth (which is equivalent to the parity of the position) is a prominent feature in the key and query embeddings (Figure 9c).

per layer. The findings observed in Figure 6b are generally consistent across training results, with the the gap between in-distribution and out-of-distribution approximation scores varying depending on the prediction type, although using lower dimensions leads to more consistent generalization gaps across categories.

Which attention head is associated with the biggest generalization gap? In Figure 17, we plot the generalization gap for each attention head in a code completion model, comparing the difference in the Same Prediction approximation score between Java examples and non-Java examples, using a rank-16 SVD approximation for the given key and query embeddings. Simplifying the fourth head in the fourth layer leads to a generalization gap for most prediction types, with the biggest gap on predicting words that appeared earlier in the sequence. In Figure 18, we plot two example attention patterns from this head. From the attention pattern, this attention head appears to implement part of an "induction head" circuit (Elhage et al., 2021). The fact that the low-rank approximation has worse prediction similarity on examples from unseen programming languages suggests that the higher dimensions of the attention embeddings play a larger role in determining the behavior of this mechanism in these out-of-domain settings. The attention head associated with the largest gap on *New word* predictions is the third head in the first layer. Two example attention patterns are illustrated in Figure 19. This attention head generally attends to the previous token, perhaps indicating that these predictions depend more on local context.

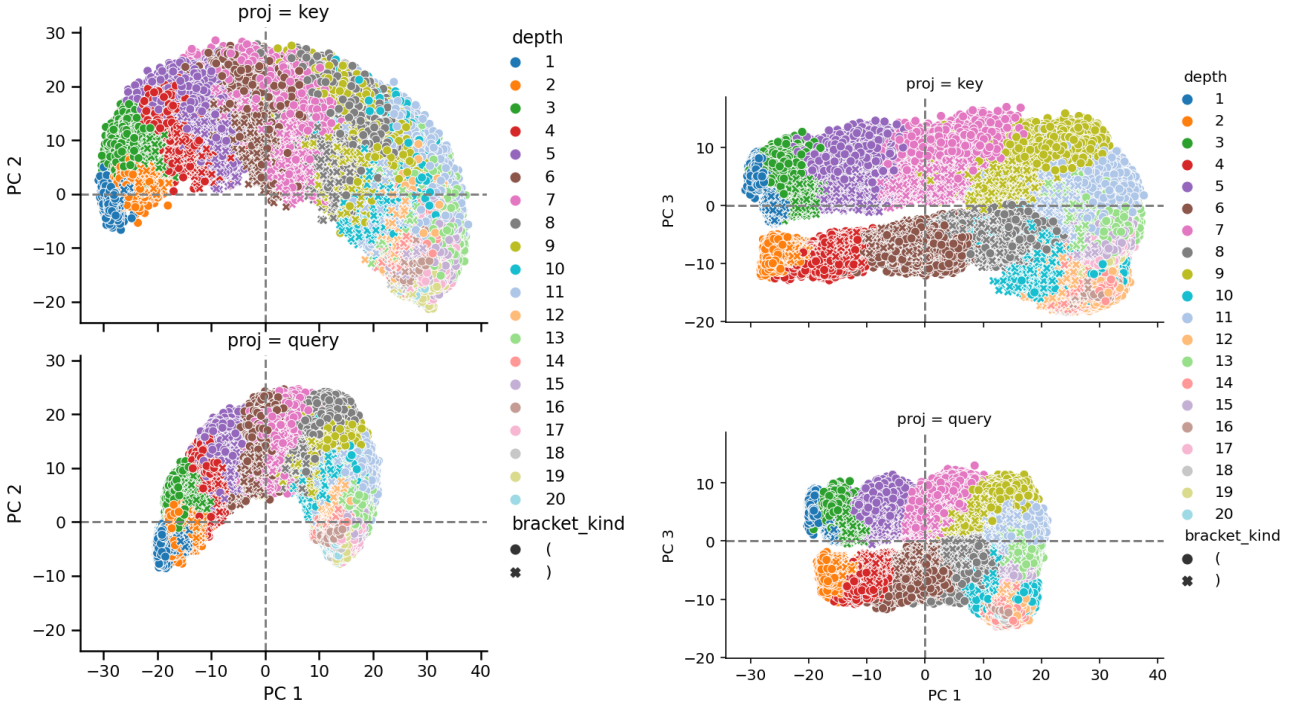
B.7. Additional Dataset: SCAN

To assess whether our findings generalize to other settings, we also train models on the SCAN dataset (Lake & Baroni, 2018). SCAN is a synthetic, sequence-to-sequence semantic parsing dataset designed to test systematic generalization. The input to the model is an instruction in semi-natural language, such as "turn thrice", and the target is to generate a sequence of executable commands ("TURN TURN TURN").

Data We consider two generalization settings. First, we test models on the **Add jump** generalization split, in which the test examples are compositions using a verb (jump) that appears in the training set only as an isolated word. We train this model on the training set introduced by Patel et al. (2022),[8] which augments the original training data with more verbs, as this was shown to enable Transformers to generalize to the *Add jump* set. Second, we test models on **Length** generalization. We use the length generalization splits from Newman et al. (2020),[9]. The examples in the

[8] https://github.com/arkilpatel/Compositional-Generalization-Seq2Seq

[9] https://github.com/bnewm0609/eos-decision



(a) First and second singular vectors.

(b) First and third singular vectors.

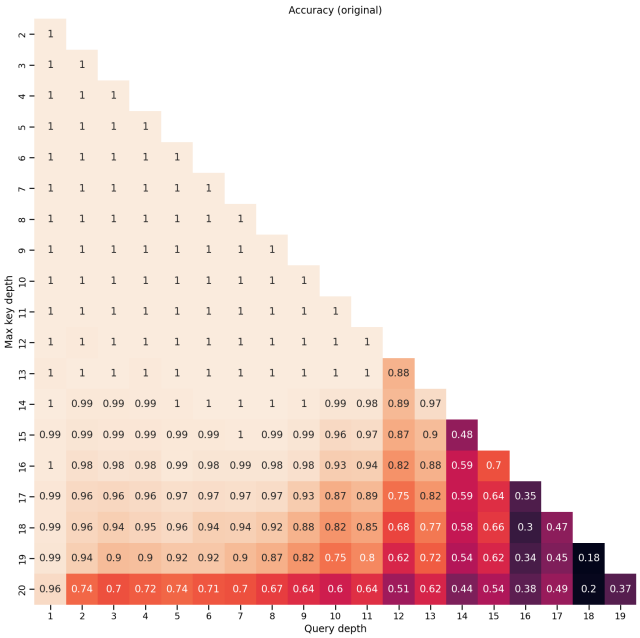
Figure 12: Key and query embeddings for sequences from the out-of-distribution depth generalization test set. Visually, the representations for out-of-distribution depths support some form of depth generalization, but the deeper depths are not separated as effectively, leading to prediction errors.

training set are no longer than 26 tokens long, and generalization splits have examples with lengths spanning from 26 tokens to 40 tokens.

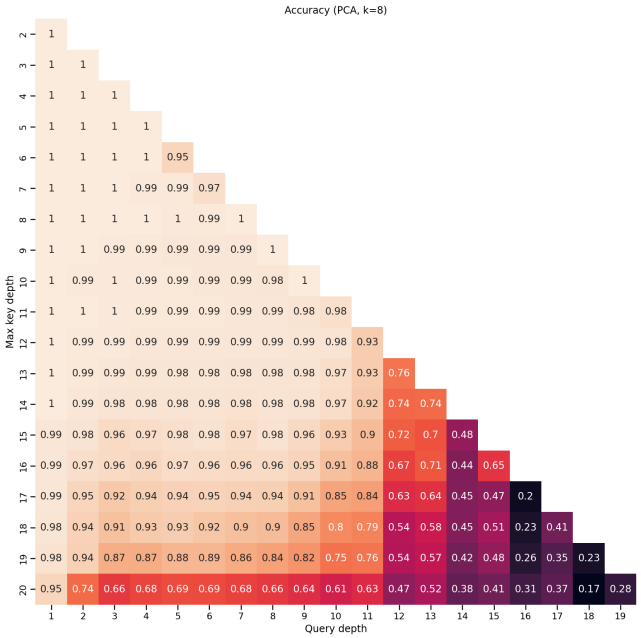
Model and training As in our Dyck experiments, we use a decoder-only Transformer. In the *Add jump* setting, we modify the attention mask to allow the model to use bidirectional attention for the input sequence, and we train the model to generate the target tokens. In the *Length* generalization setting, we use an autoregressive attention mask at all positions, and no position embeddings, because this has been shown to improve the abilities of Transformers to generalize to unseen lengths (Kazemnejad et al., 2023). We conduct a hyper-parameter search over number of layers (in $\{2, 3, 4, 6\}$) and number of attention heads (in $\{1, 2, 4\}$) and select the model with the highest accuracy on the generalization split, as our main goal is to evaluate simplified model representations in settings where the underlying model exhibits some degree of systematic generalization. The hidden dimension is 32, and the dimension of the attention embeddings is 32 divided by the muber of attention heads. For *Add jump*, this is a three layer model with two attention heads, which achieves an OOD accuracy of around 99%. For *Length*, this is a six layer model with four attention heads, achieving an OOD accuracy of around 60%. We use a batch size of 128, a maximum learning rate of 5e-4, and a dropout rate of 0.1, and train for 100,000 steps. All other model and training details are the same as in Appendix A.2 and A.3.

Results After training the models, we evaluate the effect of dimension-reduction and clustering, applied to the key and query embeddings. We apply this simplification independently to each head in the model and report the average results in Figure 20. On the *Add jump* generalization set, there is no discernible generalization gap. This result is in line with a finding of Patel et al. (2022), who found that, in models that generalize well, the embedding for the *jump* token is clustered with the embeddings for the other verbs, which would suggest that simplifications calculated from training data would also characterize the model behavior on the *Add jump* data. On the *Length* generalization set,

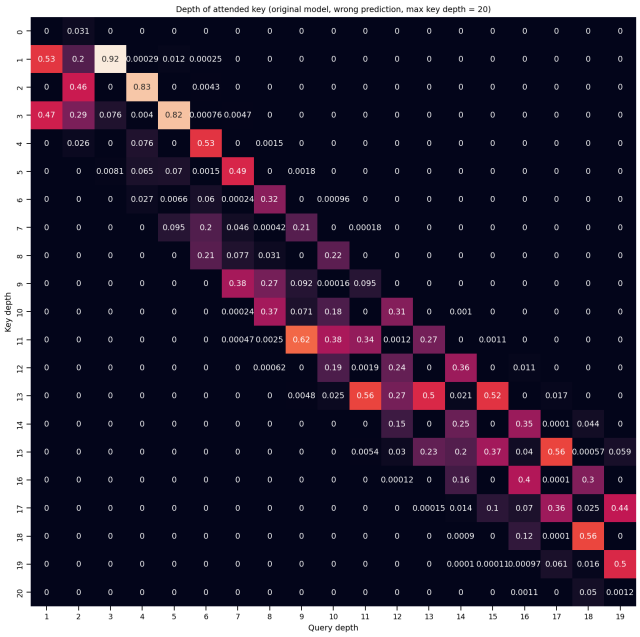
there is a small but consistent generalization gap for both dimension reduction and clustering. These results suggest that generalization gaps can also appear in other settings, and underscore the value of evaluating simplifications on a variety of distribution shifts. On the other hand, we do not attempt to conduct a mechanistic interpretation of these larger models; further investigations are needed to understand the implications of these (small) generalization gaps for model interpretation.



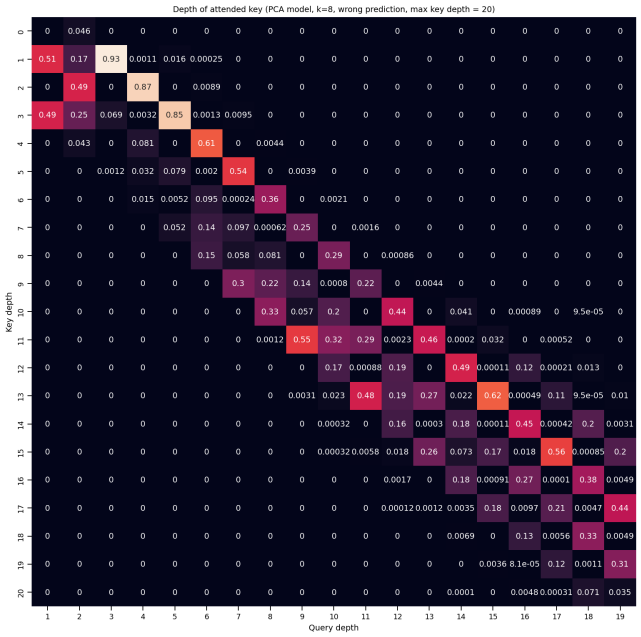
(a) Prediction errors (original).



(b) Prediction errors (SVD).



(c) Attention errors (original).



(d) Attention errors (SVD).

Figure 13: We trained a model on the Dyck dataset with a different random initialization and recreate the plots from Figure 5c. The figure plots the errors of the original model and a rank-8 SVD simplification on the depth generalization test set. Figures 13a and 13b plot the prediction accuracy, broken down by the depth of the query and the maximum depth among the keys. Figures 13c and 13d plot the depth of the token with the highest attention score, broken down by the true target depth, considering only incorrect predictions. The results are similar to the results in Figure 5c, with the simplified model diverging more on predictions with deeper query depths. For example, when the query depth is greater than 15, the lower-dimension model is more likely to attend to keys with a depth four less than the target depth.

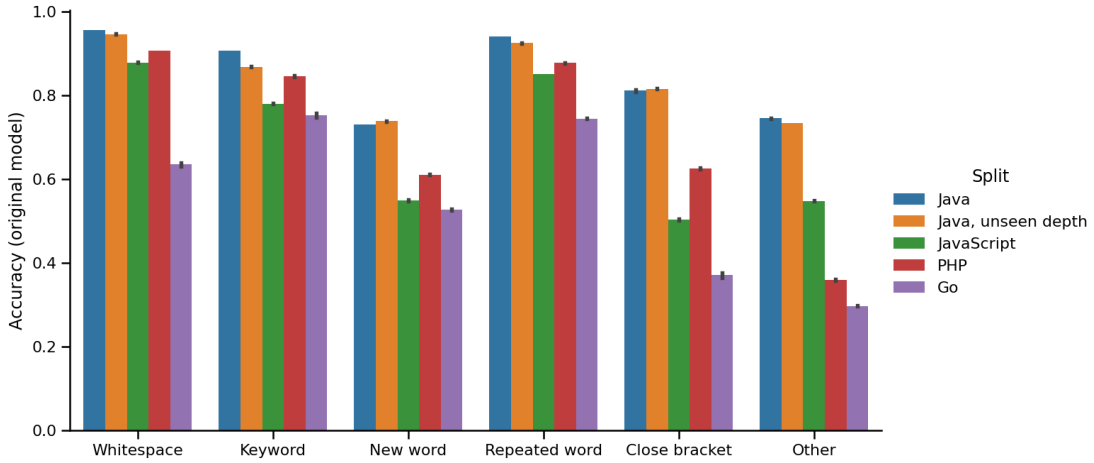


Figure 14: Accuracy at predicting the next character in CodeSearchNet, broken down by prediction type and evaluated on different generalization splits, averaged over three training runs. The models achieve relatively high accuracy, performing especially well at predicting whitespace characters and characters appearing in keywords or words that appeared earlier in the sequence. See Section B.6 for more details.

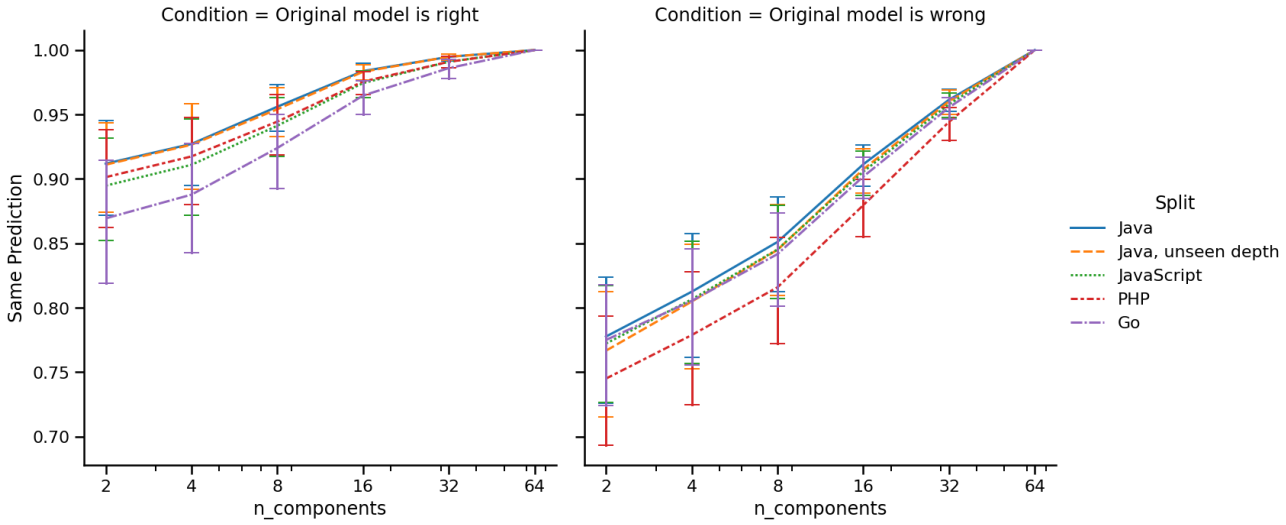


Figure 15: Predicting closing brackets in different versions of CodeSearchNet after reducing the dimension of the key and query embeddings using SVD. We apply dimensionality reduction to each attention head independently and aggregate the results over attention heads and over models trained with three random seeds. The results are partitioned according to whether the original model makes the correct (*top*) or incorrect (*bottom*) prediction. The prediction similarity between the original and simplified models is consistently higher on in-distribution examples (*Java, seen depth*) relative to examples with deeper nesting depths or unseen languages. The gap is somewhat larger when we resample the bracket types and increase the number of bracket types.

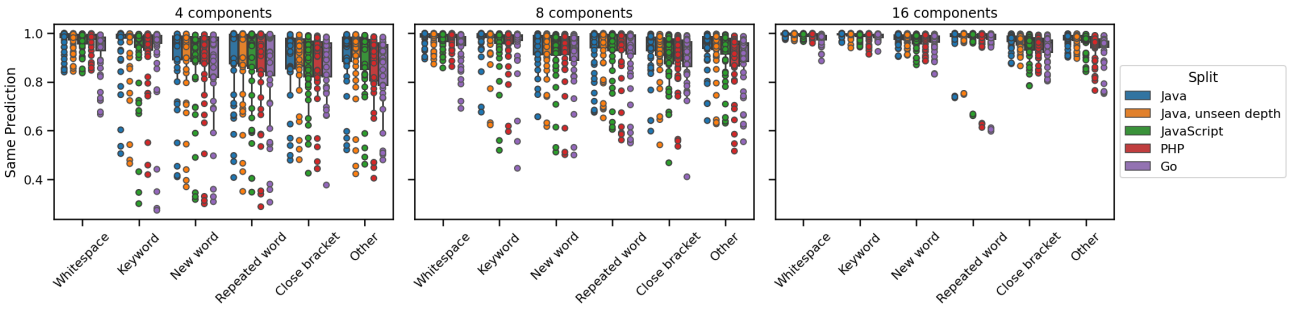


Figure 16: Prediction similarity on CodeSearchNet after reducing the dimension of the key and query embeddings using SVD, filtered to the subset of tokens that the original model predicts correctly. We train models with three random initializations. Each model has four layers, four attention heads per layer, and an attention embedding size of 64, and we apply dimensionality reduction to each attention head independently. This figure depicts the results from all three models using different numbers of SVD components, broken down by prediction type. Each point on the plot shows the prediction similarity after simplifying one attention head. The findings in 6b are generally consistent across training results, with the the gap between in-distribution and out-of-distribution approximation scores varying depending on the prediction type, although using lower dimensions leads to more consistent generalization gaps across categories.

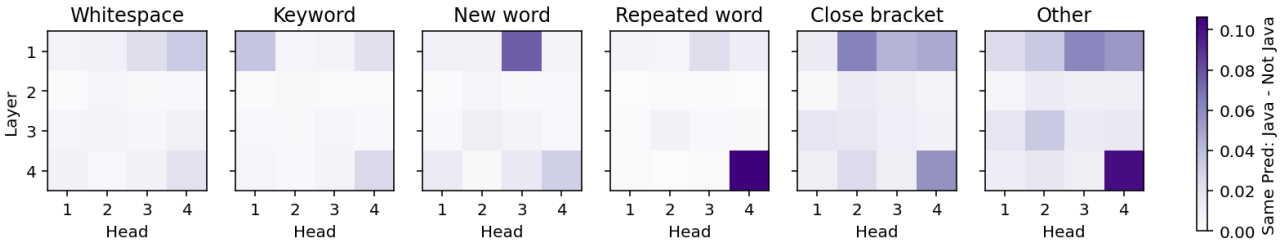


Figure 17: The generalization gap for each attention head in a model for CodeSearchNet, defined as the difference between the Same Prediction score measured on in-domain examples (Java) compared to samples in unseen languages, using a rank-16 SVD approximation for the key and query embeddings. Simplifying the fourth head in the fourth layer leads to a generalization gap for most prediction types, with the biggest gap on predicting words that appeared earlier in the sequence. This head seems to form part of an "induction head" mechanism (see Fig. 18) Simplifying the third head in the first layer leads to the largest gap on *New word* predictions; this head generally attends to token at the previous position (Fig. 19), perhaps to gather local bigram statistics. .

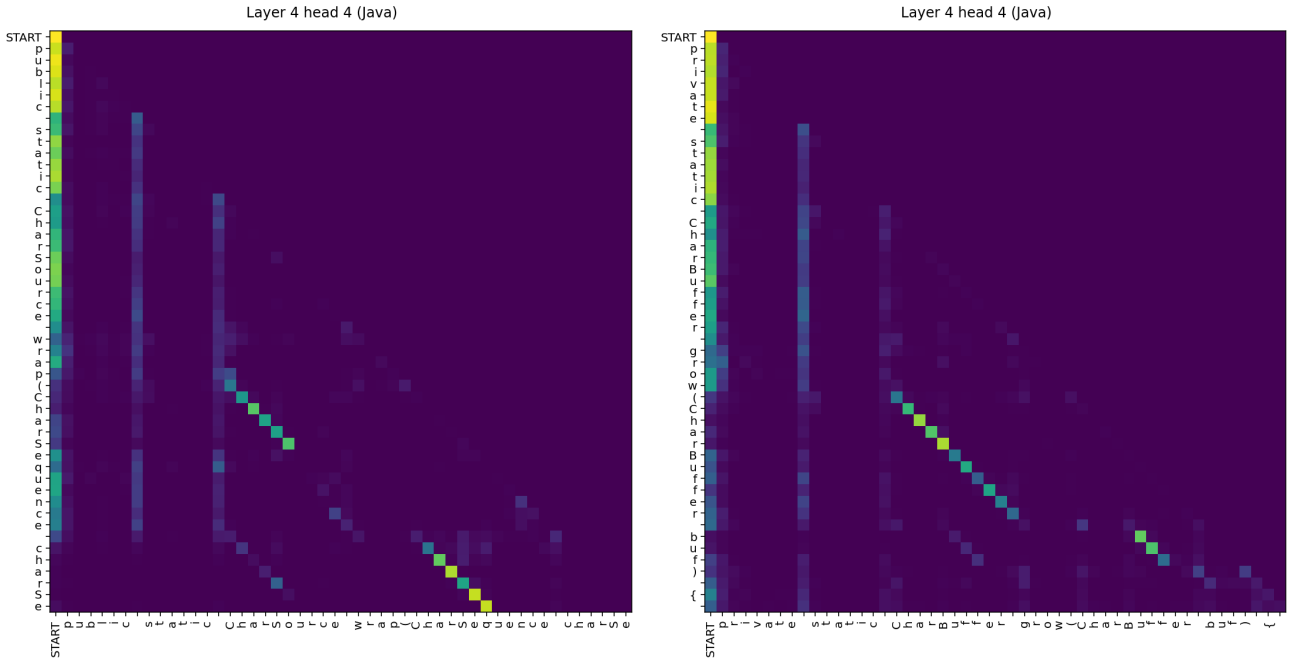


Figure 18: Two example attention patterns from the attention head associated with the largest generalization gap on *Repeated word* predictions in a code completion task (see Fig. 17). Each row represents a query and each column represents a key, and we show the first 50 characters of two Java examples. This attention head appears to implement the "induction head" pattern, which increases the likelihood of generating a word that appeared earlier in the sequence.

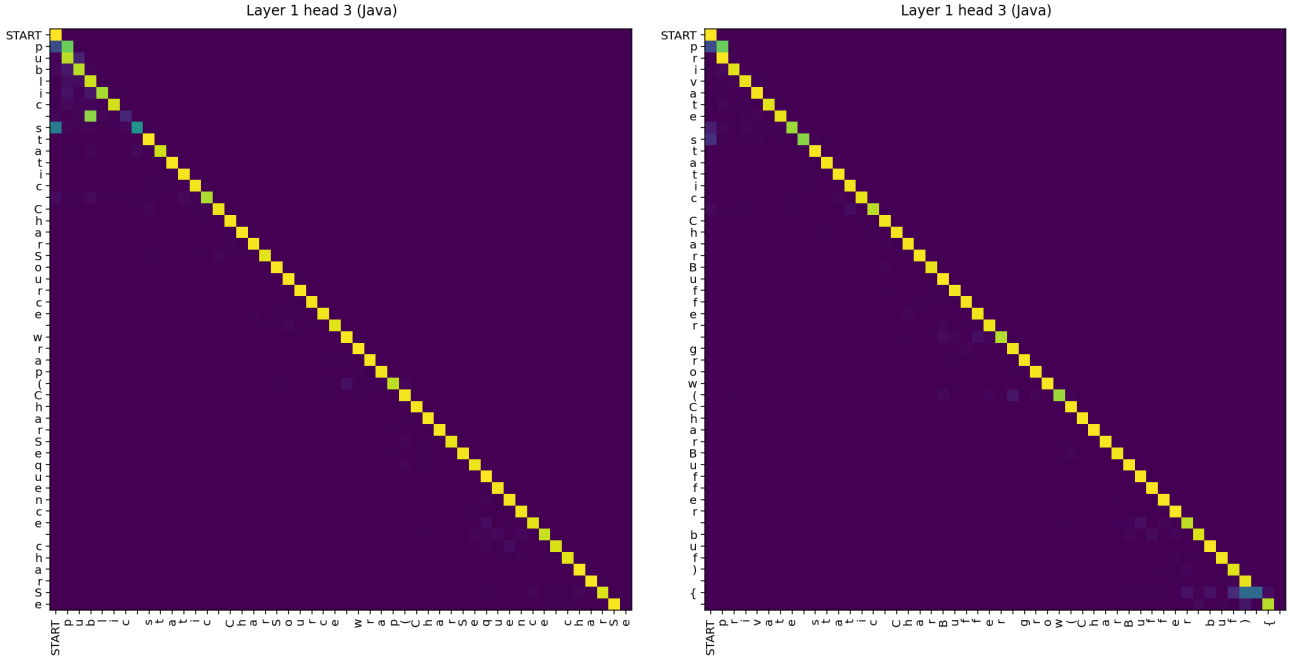
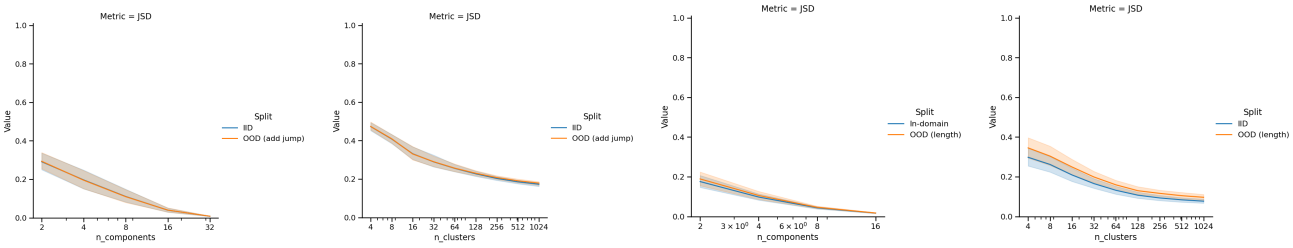


Figure 19: Two example attention patterns from the attention head associated with the largest generalization gap on *New word* predictions in a code completion task (see Fig. 17). Each row represents a query and each column represents a key, and we show the first 50 characters of two Java examples. This attention head appears to generally attend to the previous position, perhaps indicating that these predictions depend more on local context.



(a) Add jump, SVD. (b) Add jump, clustering. (c) Length, SVD. (d) Length, clustering.

Figure 20: Attention approximation metrics for models trained on the SCAN dataset, evaluated on two systematic generalization splits: the *Add jump* evaluation set, and the *Length* generalization set, which contains examples with unseen lengths. We apply these approximations to each attention head individually, and plot the results and 95% confidence interval averaged over the heads. See Appendix B.7.