

Unveiling Transformers with LEGO: a synthetic reasoning task

Yi Zhang Arturs Backurs Sébastien Bubeck

Ronen Eldan Suriya Gunasekar Tal Wagner

Microsoft Research

{zhayi, arturs.backurs, sebubeck, roneneldan, suriya.gunasekar, tal.wagner}@microsoft.com

Abstract

We propose a synthetic reasoning task, LEGO (Learning Equality and Group Operations), that encapsulates the problem of following a chain of reasoning, and we study how the Transformer architectures learn this task. We pay special attention to data effects such as pretraining (on seemingly unrelated NLP tasks) and dataset composition (e.g., differing chain length at training and test time), as well as architectural variants such as weight-tied layers or adding convolutional components. We study how the trained models eventually succeed at the task, and in particular, we manage to understand some of the attention heads as well as how the information flows in the network. In particular, we have identified a novel *association* pattern that globally attends only to identical tokens. Based on these observations we propose a hypothesis that here pretraining helps for LEGO tasks due to certain structured attention patterns, and we experimentally verify this hypothesis. We also observe that in some data regime the trained transformer finds “shortcut” solutions to follow the chain of reasoning, which impedes the model’s robustness, and moreover we propose ways to prevent it. Motivated by our findings on structured attention patterns, we propose the LEGO attention module, a drop-in replacement for vanilla attention heads. This architectural change significantly reduces Flops and maintains or even *improves* the model’s performance at large-scale pretraining.

1 Introduction

The deep learning revolution is about training large neural networks on vast amount of data. The first field transformed by this methodology was computer vision, crucially leveraging the convolutional neural network architecture [LBD⁺89, KSH12]. More recently natural language processing was revolutionized by the Transformer architecture [VSP⁺17]. Transformers are designed to process input represented as “set of elements” (e.g., the words in a sentence with their positional encoding). This is of course an incredibly generic assumption, and thus Transformers can be applied to a wide variety of tasks, including vision [DBK⁺21], reinforcement learning [CLR⁺21], and protein structure prediction [RMS⁺21, JEP⁺21] among others, or even jointly across domains to produce generalized agents [RZP⁺22]. In fact, learning with Transformers is rapidly becoming the norm in deep learning.

Transformer models display excellent performance on the standard criterion “training error/test error” (e.g., for masked language prediction or translation). However, what makes them particularly noteworthy, is that large-scale Transformer models seem to exhibit unexpected emergent behaviors, such as basic reasoning ability [TDFH⁺22, BMR⁺20, CND⁺22, DHD⁺21, RBC⁺21, HBM⁺22, SPN⁺22, ZRG⁺22, WWS⁺22,

NAGA⁺²²], excellent fine-tuning performance [HSW⁺²², TDFH⁺²², NAGA⁺²², RBC⁺²¹, PHZ⁺²²], or zero-shot learning [BMR⁺²⁰, CND⁺²², DHD⁺²¹, RBC⁺²¹, HBM⁺²², SPN⁺²², ZRG⁺²²]. Currently, there is a remarkable community effort towards *at-scale* experimental investigation of Transformers, essentially trying to find out what such models can do when they become large enough and are trained on large/diverse enough datasets. The successes are striking and capture the imagination [BMR⁺²⁰, RDN⁺²²]. Yet, for all of these wonders, there is very little *understanding* of how these models learn, or in fact what *do* they learn. Answering such questions in the *at-scale* experiments is particularly challenging, as one has little control over the data when hundreds of billions of tokens are harvested from various sources. In this paper, we propose to take a step back, and try to understand how learning occurs and what is being learned in a more controlled setting that captures important aspects of “reasoning”.

The benefit of such a controlled setting is that we can try to understand some of the most pressing questions in learning with Transformers, particularly around (i) the architecture and (ii) the importance of training data. For (i) we probe the role of multiple heads and depth, and we show that we can successfully understand them in our controlled setting. For (ii) we investigate how much the dataset composition matters, as well as how pretraining on merely vaguely related tasks makes fine-tuning successful. In turn, these insights can guide our thinking for large-scale experiments, and we give some of the lessons learned below. In particular, our insights crystallize into an architectural change to BERT for faster inference with matching or even better performance (Section 5).

1.1 LEGO: A synthetic reasoning task

Core components of reasoning include the ability to *associate* concepts, and to *manipulate* them. We propose a simple task that captures these two aspects, which we call LEGO (Learning Equality and Group Operations). In LEGO, the input describes a sequence of *variable assignments* as well as *operations* on these variables by a fixed (mathematical) group. One needs to be able to deal with both long-range assignments (the same variable appearing in different parts of the input should be viewed as a being *equal* to same quantity), as well as short-range operations (describing what group element is applied to which variable). A key parameter of an input sequence will be its length, which is proportional to the number of sequential reasoning steps one has to do in order to resolve the value of each variable. We will mostly train with a fixed sequence length (say 12). We often provide supervision only on part of the sequence (say the first 6 variables). We do so in order to test the generalization capabilities from smaller length sequences to longer length sequences without introducing potential errors due to the positional encoding in Transformers.

1.2 Some takeaways

In LEGO, we are interested in both *classical generalization* (i.e., training and test distribution are the same) and *out-of-distribution generalization*. For the latter we focus on distribution shifts that vary the length of the chain of reasoning, and thus we refer to this type of generalization as *length extrapolation*. Specifically, the setting for length extrapolation is to train with supervision on shorter sequence lengths (e.g., supervision on only the first 6 variables) and test on a long sequences (e.g., accuracy computed on 12 variables). A summary of our empirical observations is as follows:

1. First, classical generalization happens reliably for all architectures and data regimes.
2. More interestingly, length extrapolation seems to depend on architectural/data composition choices. Specifically, BERT-like models without special data preparation do *not* extrapolate to longer sequences, while other models like ALBERT, or BERT with carefully selected data (such as diverse sequence lengths, or pre-trained BERT) *do* extrapolate.
3. The extrapolating models all seem to evolve attention heads dedicated to either *association* (long-range identity matching) or *manipulation* (short-range operations). We provide evidence that pre-trained BERT

(which is pre-trained on a seemingly unrelated dataset) generalizes because it has learned such heads.

4. The non-extrapolating models seem to solve the classical generalization problem using a certain shortcut-like solution, whereby using the specificity of the group operations they are able to jump to the end of the chain of reasoning, and then complete the rest of the variables by following the reasoning both from the start *and* the end of the chain.

We interpret our findings as follows:

- (i) Classical generalization can be a deceptive metric, as there might be unexpected ways to solve the problem. This is famously related to the issue of embedding machine learning systems with *common sense reasoning*. Namely, we hope that when an ML system solves a task, it does so in “the way humans do it”, but of course, nothing guarantees that this will happen. Our findings are consistent with the current methodology of increasing the diversity of the training data, which seems crucial for generalization.
- (ii) ALBERT-like models, where a layer is repeated several times, seem to be an ideal structure for problems that could be described algorithmically as a “for loop” (as is the case with following a chain of reasoning). Indeed we find that ALBERT extrapolates in data regimes where BERT does not, clearly separating these two architectures.
- (iii) The success of pretraining/fine-tuning in vastly different tasks might actually come from a “simple” better initialization, rather than complex knowledge encoded during pre-training.
- (iv) The interplay between short-range (close-by information in a sentence) and long-range (the same concept appearing in different places in the sentence) is relevant more broadly than in our synthetic task. We observe that the networks effectively learn to deal with short-range/long-range information by implementing specific attention patterns. This motivates us to study a new LEGO attention architecture, and we show it matches or even outperforms its baseline on the large-scale pretraining but with significantly less computational cost.

1.3 Related works

In [ZRKB21], the PVR (Pointer Value Retrieval) task is introduced, with a similar high-level goal to ours in introducing the LEGO task, namely to study how neural networks learn to reason in a controlled setting. In a PVR task, part of the input indicates another part of the input where a function of potentially varying complexity has to be computed. Like us, they use distribution shift to investigate how various network architectures learn this task, and they observe that networks can learn the task at hand (“classical generalization”) yet fail to extrapolate to mild distribution shift. They then ask the following questions: “Are there architectural changes that can enforce better priors and withstand distribution shift? Can novel learning objectives prevent these adversarial correlations? Progress on these questions holds promise for greater robustness.”

Our study attacks these questions directly in the context of the LEGO task (e.g., ALBERT versus BERT, and training set composition investigations), and our preliminary results indicate that this is indeed a fruitful direction to obtain better models in some aspects (e.g., more interpretable). Other examples of recent synthetic benchmark with a similar philosophy include SCAN (Simplified version of the CommAI Navigation) [LB18], CFQ (Compositional Freebase Questions) [KSS⁺20], LIME [WRL⁺21], PCFG SET [HDMB20], and BONGARD-LOGO [NYM⁺20]. In SCAN for example, one has to “translate” a command of the form “turn left twice and jump” into a sequence of actions “LTURN LTURN JUMP” (see [PBBG22] for more recent progress on this dataset). Again, similarly to the PVR tasks, these works focus on understanding generalization (in these cases, *compositional generalization*). Another related line of works is on studying Transformers to recognize various formal languages, see e.g., [BAG20, YPPN21]. A contemporary work [CIS21] proposed modifications to Transformer architectures to achieve significantly better length extrapolation (other works

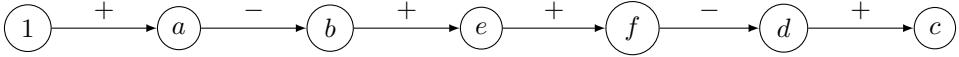


Figure 1: The graph representation of the sentence $a = +1; b = -a; e = +b; d = -f; c = +d; f = +e$

studying this important class of distribution shifts include [AWA⁺22]). As far as we know, none of these works try to probe the inner workings of the networks in the same depth as we do here. On the other hand, networks trained on real data are being extensively scrutinized, see for example [RKR20] where they try to understand some of the attention heads of BERT (see also [SGSB20a]). However, making sense of these real-data-trained networks is a daunting task, and a key contribution of ours is to show in a limited setting one can obtain a clearer picture of what Transformers learn.

The LEGO task is also naturally related to the growing literature on testing mathematical/coding abilities of Transformers (e.g., [SGSB20b]), specifically the simpler tasks of checking the correctness of a proof (or simplifying one, such as in [AAG21] which studies simplification of polynomials), or executing code for a given input [CST21]. It would be interesting to see if some of the insights we derive in the present paper apply to currently challenging mathematical tasks such as MATH [HBK⁺21] and IsarStep [LYWP21].

There are an abundance of studies on attention heads that have identified the importance of local, convolutional, attention patterns [VTM⁺19, CNM19, CKLM19, RST20, YSI20]. However, to the best of our knowledge, we are the first to demonstrate the importance of the association pattern that globally attends to identical tokens, thanks to the LEGO task.

2 Learning equality and group operations (LEGO)

We propose the following synthetic task, which we call LEGO. Let G be a finite (semi)group acting on a finite set X , and denote $g(x)$ for the action of $g \in G$ on $x \in X$. We define a formal language using the symbols from G and X as well as symbols from a finite alphabet A which we refer to as the *variables*. A sentence in our formal language is made of clauses separated by a semi-colon. A clause is of the form $a = gx$ with $a \in A$, $g \in G$ and either $x \in X$ or $x \in A$. If $x \in X$, such a clause means that the variable a is assigned the element $g(x) \in X$. On the other hand if $x \in A$ and the variable x was assigned an element $y \in X$ through another clause (or chain of clauses) in the sentence, then the clause $a = gx$ assigns variable a to the element $g(y) \in X$. The task’s goal is to take in input a sentence with a fixed number n of clauses, given in an arbitrary order, and to output the assigned element to each variable that appear in the sentence (the formal language will have a further restriction that ensures that each variable is assigned one and only one element).

We can view a sentence as a directed graph on the vertex set $X \cup A$ with labelled edges as follows: a clause $a = gx$ corresponds to a directed edge from the vertex x to the vertex a , and the edge is labelled with g . We restrict our attention to sentences corresponding to a line graph directed away from some fixed root vertex $r \in X$, and whose non-root vertices are all in A , see Figure 1 for an example. In particular such sentences are “consistent”, meaning that a variable is assigned a unique element (the assignment is obtained by simply “following the chain”).

Task 1. The most basic instantiation of LEGO is when G is the unique group of 2 elements acting on a set X also of 2 elements, that is $G = \{+, -\}$ and $X = \{1, -1\}$. Our sentences thus consists of n clauses of the form $a_i = \pm a_{i-1}$, where $a_i \in A$ for $i = 1, 2, \dots, n$ and $a_0 = 1$ (we fix $r = 1$). Note that in this case our formal language has well over a billion unique valid sentences when $n \geq 10$. Example of a sentence with $n = 6$ is (see Figure 1 for the graph depiction): $a = +1; b = -a; e = +b; d = -f; c = +d; f = +e$. Our task’s goal is to report the elements or values from X assigned to the variables appearing in the sentence. In the above example, assignments for variables a, b, c, d, e, f are $1, -1, -1, -1, 1, 1$.

Task 2. One can think of Task 1 as the case of LEGO for the permutation group on $N = 2$ elements (acting on itself). Our second task will correspond to $N = 3$, which is qualitatively different since the permutation group on 3 elements is non-abelian.

We will focus on Task 1 in the main paper and include in Appendix E experiments on this Task 2. Our training and test data for the task consists of n length chains as described above with the order of clauses in the sentence randomized. A sample input sentence to a transformer looks like [BOS] $j=-f$; $f=-b$; $y=+t$; $o=+e$; $d=+y$; $v=+d$; $h=-o$; $b=-i$; $i=+1$; $t=+l$; $e=-j$; $l=-h$; [EOS]. See appendix for further data generation details.

3 Transformers for LEGO

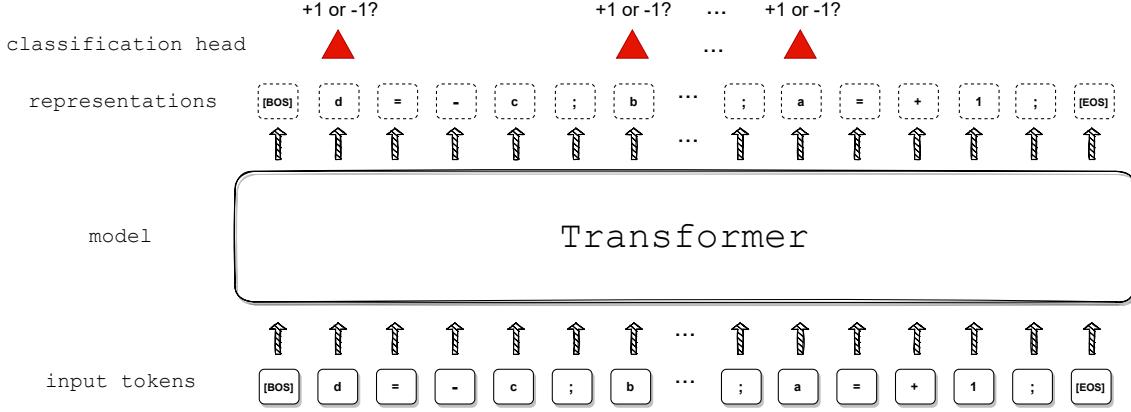


Figure 2: Illustration of a transformer model applied to LEGO Task 1 on input sentence $d=-c$; $b=-a$; $c=+b$; $a=+1$. We apply a linear classification head to the output representations of each clause’s first token to generate predictions for the variables assignment.

We apply transformer models in the token classification pipeline to predict the assignments of the variables in the input sentence, depicted in Figure 2. To evaluate the out-of-distribution generalization (referred to simply as generalization), we introduce the notation of $n_{tr} \leq n$, such that during training, supervision is provided only on the first n_{tr} clauses (first in the graph representation of the input sentence). We mainly focus on BERT [DCLT18] and ALBERT [LCG⁺19] architectures. These two models are representative large transformer architectures for NLP tasks, and we observe they exhibit intriguing behavior difference on our tasks which we will detail in Section 4. See appendix for training hyper-parameters and dataset construction details.

In Figure 3, we report initial results on LEGO with $n = 12$ and $n_{tr} = 6, 12$. Both BERT and ALBERT are able to achieve good classical generalization, while only ALBERT appears to generalize even to slightly longer sequence length. We observe similar behavior across different lengths of inputs too. This suggests that classical generalization might be a deceptive metric to evaluate learning of true logic/reasoning tasks. Motivated by these initial results, in the next section we focus on breaking down the learning dynamics of BERT and ALBERT for the LEGO task towards carefully understanding their strengths and weaknesses.

4 Unveiling Transformers with LEGO

4.1 BERT vs. ALBERT: Iterative reasoning in iterative architectures

A salient feature of many reasoning tasks is an iterative component, meaning they can (or must) be solved by sequentially repeating certain operations. In this section, we use LEGO to study and compare Transformer architectures through the lens of iterative reasoning.

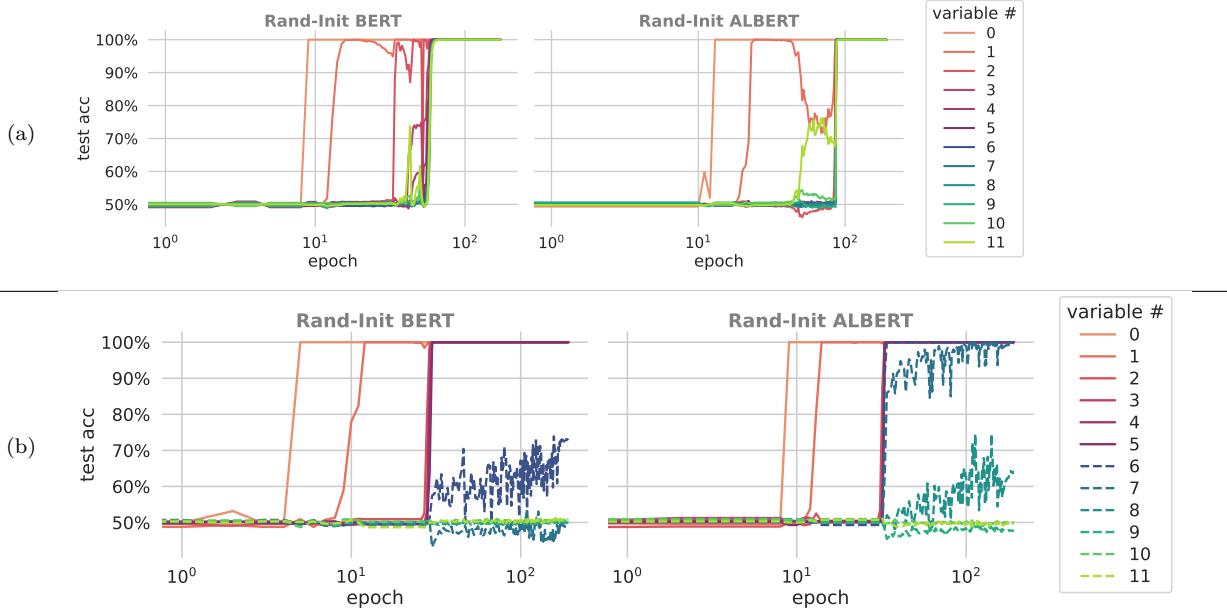


Figure 3: Solving LEGO (Task 1) using BERT and ALBERT, trained from random initialization. Each curve corresponds to the test accuracy of a single variable appearing in the sentence over the course of training. The variable numbers in the legend are their position in the reasoning chain (or graph representation) of the input sentence, rather than the position in the sentence itself. For example, on the input sentence: $b=-a; d=-c; c=+b; a=+1;$, variable #0 is a , #1 is b , #2 is c , and #3 is d . Top a): models are trained to fit all variables ($n = 12, n_{tr} = 12$). Bottom b): models are trained to fit the first 6 variables but test on all 12 variables ($n = 12, n_{tr} = 6$). Dashed curves represent variables not supervised during training.

A natural solution to LEGO—and arguably the go-to solution for a human—is to implement a “for-loop”, where each iteration resolves one step in the reasoning chain. The iteration could look for the next unresolved variable token whose value could be resolved in one step. Iterative Transformer architectures such as ALBERT and Universal Transformers [DGV⁺18], where the weights are shared across different layers, inherently implement a for-loop with a number of iterations equal to the number of layers. If the model manages to learn to implement one such iteration during training, the network would immediately be capable of performing length extrapolation. If this indeed occurs, it would point to a clear advantage of ALBERT over BERT in our setting. This leads to the following questions.

Q1. Do iterative architectures indeed exhibit better length extrapolation?

The bottom plots of Figure 3 display the length extrapolation result for BERT and for ALBERT. They show the clear advantage of recurrence: While the non-iterative BERT achieves only somewhat better-than-random accuracy for one variable (#6) beyond the ones accounted for during training (#0- #5), the iterative ALBERT reaches near-perfect accuracy on two additional variables (#6 and #7), and nontrivial accuracy on the third (#8). These results clearly support that iterative architectures do generalize better in the iterative LEGO reasoning task.

Q2. Does the ALBERT architecture actually implement the for-loop?

To a lesser extent, Figure 3 also hints at a positive answer to Q2. Observe that ALBERT exhibits length extrapolation to variable #6 immediately (in terms of epochs) as soon as it fits the training variables (#0

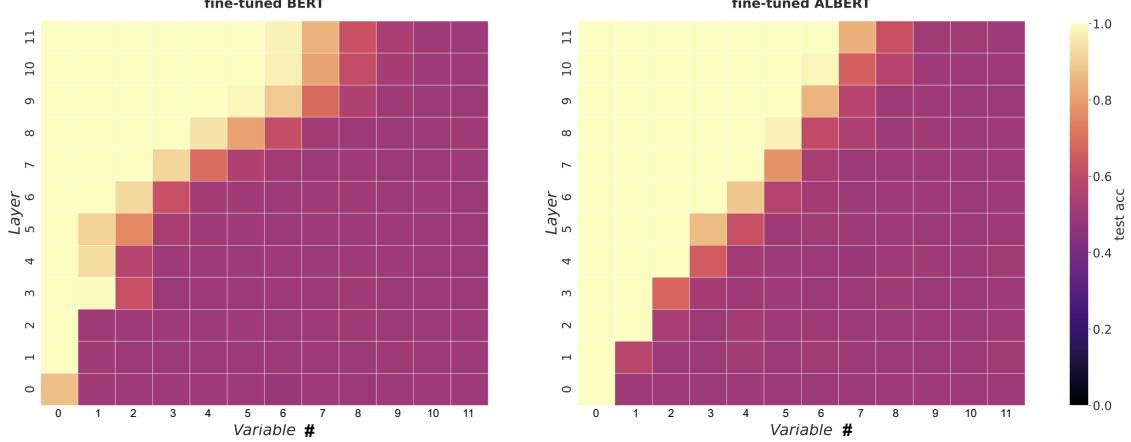


Figure 4: Visualization of information percolation within the fine-tuned models. The color indicates the test accuracy of the probing classifier at each layer. Brighter is higher. We observe ALBERT’s information percolation is linear than BERT’s, which implies ALBERT is biased towards learning a for-loop.

– #5), whereas for BERT, the corresponding plot (#6) climbs gradually even after the training variables are predicted perfectly. This suggests that once it manages to learn the operations required for one step of reasoning, it can immediately implement those operations over a few more iterations not required in training.

In order to gain stronger evidence, we measure the dependence between the location of a variable token in the chain and the layer in which its value is typically resolved. To this end, given a trained model, we train one linear classifier per layer which predicts the value of a variable token based only on its token representation at the corresponding layer (without using other information), while keeping the original model unchanged. This allows us to gauge the rate of information percolation along the reasoning chain in terms of layers per reasoning step. If the model indeed implements a for-loop in its forward pass, one expects a linear relationship between the number of layers and the number of reasoning steps already completed. We visualize in Figure 4 the test accuracy of prediction as a function of the layer in the network and depth in the chain. While not perfectly linear, the relation clearly looks closer to linear in ALBERT, suggesting that the ALBERT model has an inductive bias towards learning to implement the “natural” for-loop with its forward pass.

Q3. How can we incentivize models to learn iterative solutions?

We attempt to incentivize the model to implement the “natural” for-loop solution. We rely on the observation that if each iteration of the for-loop simply percolates the information one more step (assigning a value to the next variable), then adding more layers with the same weights should not affect the output, and in fact, one should be able to read out the output of the calculation from any layer of the neural network, as long as its depth exceeds the length of the chain. With this observation in mind, we train a ALBERT model with stochastic depth [HSL⁺16]. We uniformly sample depth between 6 and 12 per batch during training while fixing it at 12 during test. Figure 5 shows a clear improvement in generalization to longer lengths using stochastic depth.

4.2 Rand-Init vs. Pretrained: Structural advantages from pretraining

Pretraining large models has emerged as a prominent and highly successful paradigm in large-scale deep learning. It advocates first training the model on a large dataset to perform a generic task, followed by task-specific fine-tuning on the task at hand. Our goal here is to use LEGO as a testing ground for this

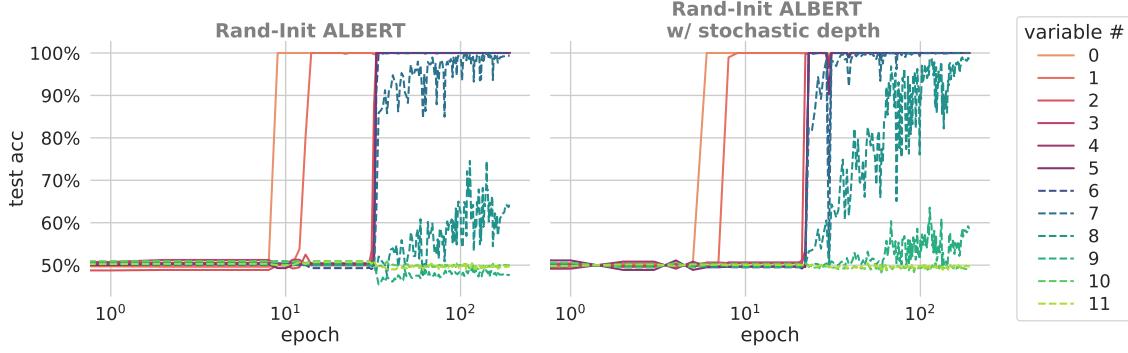


Figure 5: Generalization of ALBERT trained with stochastic depth. The stochastic depth improves the length extrapolation to longer sequence lengths.

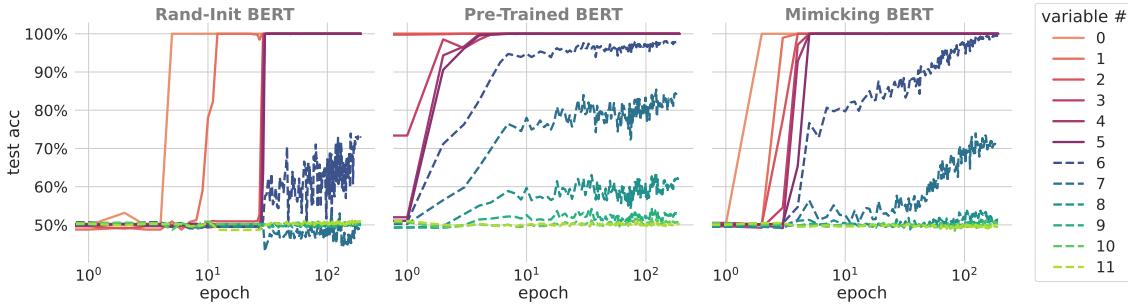


Figure 6: Pretrained BERT exhibits significant performance advantages over its Rand-Init counterpart, while the mimicking procedure (a simple initialization scheme we describe below) heads closes the gap.

paradigm. To this end, we compare (a) training the BERT architecture for LEGO from random initializations to (b) fine-tuning the standard pre-trained BERT model to solve LEGO. Figure 6 (left and center plots) shows that pretraining helps generalization in LEGO dramatically: the pre-trained model generalizes to unseen sequence lengths (the dashed plots) much better, and within a far smaller number of epochs, than the randomly initialized model.

4.2.1 Why does pretraining help in LEGO?

One simple explanation is that pre-trained BERT is already aware of the semantics of tokens like ‘=’ or ‘-’. We have easily ruled out this possibility, by replacing those tokens with arbitrary ones that do not encompass the same semantics; this does not affect the performance of pre-trained BERT. A more intriguing explanation pertains to the attention mechanism itself. At its basis, LEGO requires two fundamental types of information transfer:

- **Association:** encoding long-range dependencies that transfer a value between two occurrences of the same variable. For example, if the input contains the two clauses “ $a = +1$ ” and “ $b = -a$ ” (with arbitrary separation between them), the architecture must associate the two occurrences of the variable a in order to correctly set b to -1 .
- **Manipulation:** encoding short-range dependencies of transferring a value from the right-hand to the left-hand side of the clause. For example, to successfully process the clause “ $b = -a$ ”, the architecture must associate these particular occurrences of a and b with each other, in order to transfer the value of a (after applying to it the group element -1) into b .

Association corresponds to a purely global attention pattern, completely reliant on the *identity* or *content*

of the tokens and oblivious to their *positions* in the input sequence. Manipulation, in contrast, corresponds to a purely local attention pattern, where nearby positions attend to each other.

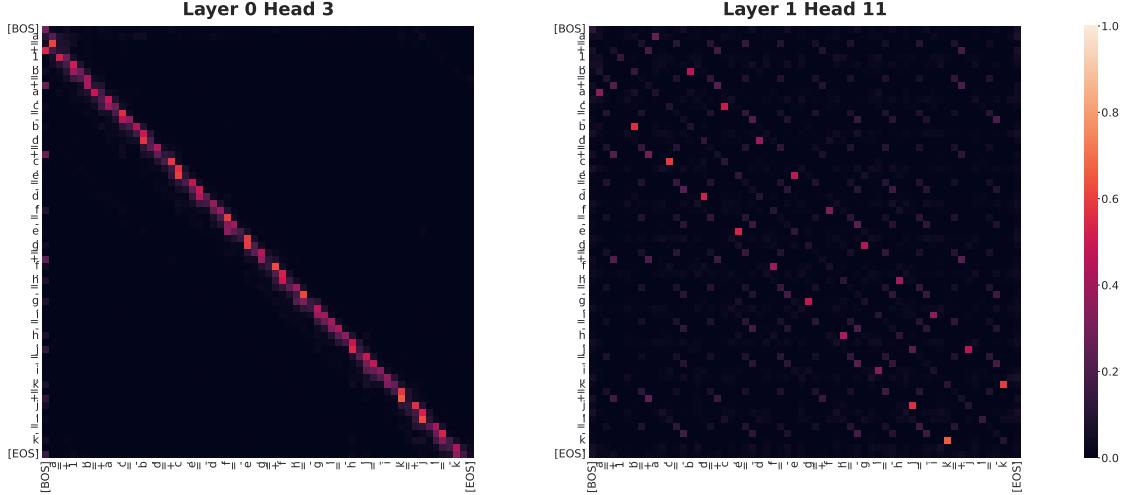


Figure 7: Visualization of two representative attention maps from a pre-trained BERT model not yet fine-tuned on LEGO. A complete visualization of all attention patterns of the pre-trained BERT is in Appendix F. On the LEGO input sequence, certain heads implement local, convolution-like manipulation operators (left), while some others implement global, long-range association operators (right). Note that the sample input sequence is presented in the reasoning chain order for visualization purposes only.

It is natural to ask whether they are indeed manifested in the pre-trained model’s attention heads in practice. Indeed, Fig. 7 shows two exemplar attention heads of pre-trained BERT on an input LEGO sequence without any fine-tuning. The right head clearly depicts association: each token attends to all other occurrences of the same token in the input sequence. This motivates us to make the following hypothesis: ***the advantage of pre-trained models on LEGO can be largely attributed to the association and manipulation heads learned during pretraining.***

Note that merely the existence of the heads does not fully validate the hypothesis yet. To rule out other factors, we carefully design controlled experiments to test this hypothesis in the section below.

4.2.2 Verifying the hypothesis with Mimicking

To test this hypothesis, we conduct the following *mimicking* experiments.

Mimicking BERT We ‘initialize’ certain attention heads to perform association and manipulation, without access to pretraining data. We achieve this by specifying the target attention matrices (one for association and one for manipulation), and training the model on random data to minimize a “mimicking loss” that measures how well the actual attention matrices at every layer match the target matrices. The precise mimicking loss and training protocol are specified in the Appendix B.3. The rightmost plot in Figure 6 shows that BERT with mimicking initialization attains significant advantage in generalization over randomly initialized BERT, despite not being pre-trained on any real data (and thus not having learned to “reason”). This confirms that much of the advantage of pre-trained BERT stems from having learned these information transfer patterns.

4.3 Shortcut solutions and their effect on generalization

As discussed in Section 4.1, a natural solution to LEGO is to resolve variables iteratively by the order of their depth in the chain. Surprisingly, we find that the Rand-Init BERT and ALBERT models first learn a “shortcut” solution: they immediately resolve the *last* variable in the reasoning chain, perhaps by counting the total number of minus signs. Indeed, the last variable can be easily identified as it appears only once whereas every other variable appears twice, and its value is fully determined by the parity of the number of minus signs. This behavior is observed in Figure 3a: the randomly initialized models are trained to fit all 12 variables: the last variable (#11, indicated by the brightest green curves) improves earlier than *almost* all other ones.

This behavior may be related to the well-observed phenomenon of *spurious features*: a model succeeds in training not relying on any actual features of cows and circumventing the intended solution [MPL19, SHL20, GSL⁺18, NNSN21].

We use LEGO as a case study of shortcut solutions and their effect on generalization. Instead of training the model to fit the first six variables (as in bottom Figure 3 in Appendix), we train it to fit the first five (#0–#4) and the last variable (#11). This allows us to measure length extrapolation (to #5–#10) in a setting where models can learn the shortcut. The results show significantly degraded performance, implying that shortcut solutions impede generalization. We then study ways to prevent models from learning them, by pretraining and mimicking. The full section appears in Appendix A.

5 LEGO Attention: faster and better

Our analysis in Section 4.2 reveals that the advantage of the pre-trained BERT model on LEGO originates from two specific types of attention structures emerging from pre-training — the association and manipulation patterns. A quick examination of all the attention heads depicted in Appendix F suggests that there is one more clearly identifiable attention pattern: broadcasting on the [CLS] token or the [SEP] token (sometimes both). Namely, it ‘broadcasts’ the value inside the special tokens to the others. Even though [CLS], [SEP] play no role on LEGO per se, they are vital to the pretraining objective as well as many downstream tasks. Thus the broadcasting attention pattern is presumably important for many real-life NLP tasks beyond LEGO. Association, manipulation, and broadcasting consist of a considerable portion of the pre-trained BERT’s attention heads, and they are so structured that we can in fact LEGO v0 them efficiently.

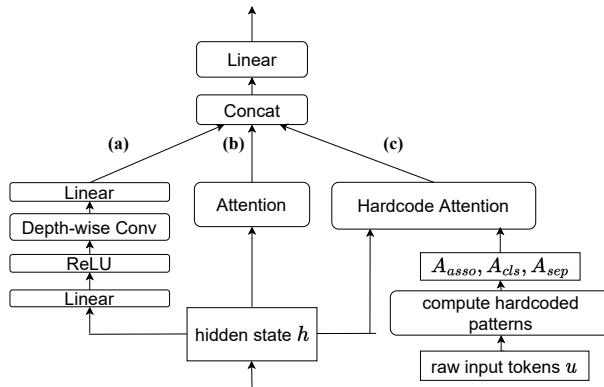


Figure 8: Our proposed LEGO attention consists of 3 pathways. BERT has pathway (b) only; the LEGO v0 attention module has (a) and (c); the LEGO v1 attention has (a), (b), and (c). See Appendix C.

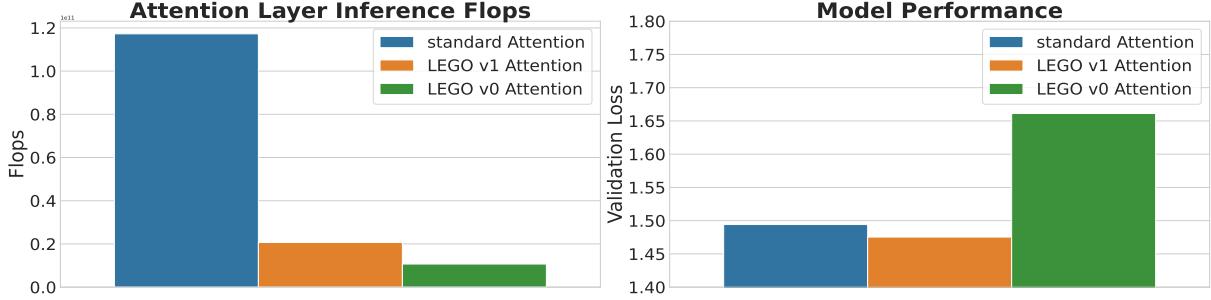


Figure 9: Top) Comparison of inference Flops and model size. Flops are measured on a batch of 64 sequences of 512 tokens.

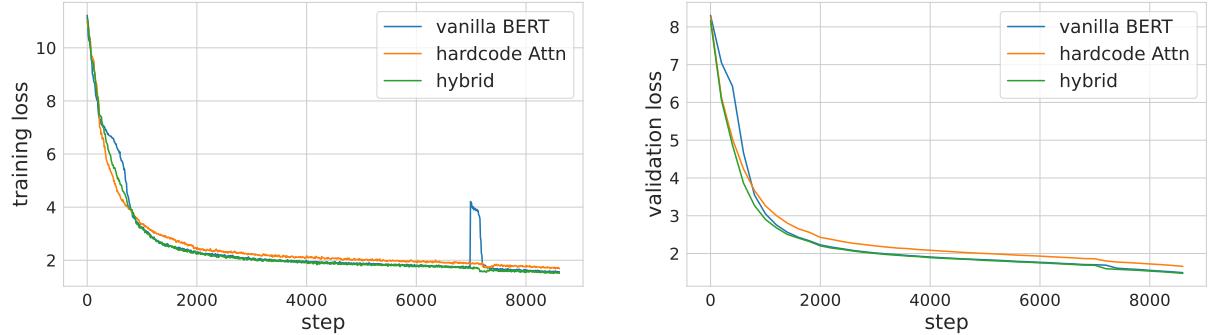


Figure 10: Training and validation performance on BERT pertaining task (Masked Language Modelling+Next Sentence Prediction). As a standard, the training sequence length increases from 128 to 512 around the 7k-th step, where the BERT training loss exhibits a sudden bump in response, while the LEGO v0/v1 models exhibit remarkable resilience. The LEGO v1 model learns faster and (slightly) outperforms BERT in validation.

LEGO Attention: For the association, manipulation, and broadcasting heads, we can efficiently construct the sparse attention matrix based on the input token IDs only, without learning Q and K or the expensive attention probability computation. For manipulation maps, due to their intrinsic locality, we decide to implement them directly with temporal convolutions (along the time dimension). For the other global maps, given a *raw* input sequence of T tokens, $u_1, u_2, \dots, u_T \in \mathbb{N}$, we manually construct the association and broadcasting maps $A_{asso}, A_{cls}, A_{sep} \in \mathbb{R}^{T \times T}$ such that $(A_{asso})_{ij} = \mathbf{1}[u_i = u_j]$, $(A_{cls})_{ij} = \mathbf{1}[u_j = [\text{CLS}]]$, $(A_{sep})_{ij} = \mathbf{1}[u_j = [\text{SEP}]]$ where $\mathbf{1}[\cdot]$ is the indicator function which outputs 1 if the argument is true and 0 otherwise. In the end, we normalize them to have row-wise unit ℓ_1 norm. Notably, the latter three patterns require no training (except for a value map for each layer) and are shared across all layers.

On the standard BERT pertaining benchmark, we compare the following three models: BERT-base model, LEGO v0 and v1 models. We use convolutional kernel size 21 for the latter two. In Figure 10, we show that the LEGO v0 model learns fast in the beginning but falls short later on. However, the LEGO v1 model not only reduces model size and accelerates inference, but also renders models that are extremely competitive with the base model in terms of the final performance of large-scale pertaining. We follow precisely the training pipeline and hyperparameters of [DCLT18]. See Appendix C for architecture details of the LEGO v0/v1 models.

We observe that the LEGO v0 model learns faster but gradually falls short, while the LEGO v1 model achieves the best of both worlds: it learns faster at the beginning and achieves even (slightly) lower validation loss at the end. The LEGO v1 model’s validation loss curve appears to be a lower envelope of the other two. The BERT/LEGO v0/v1 models achieve 1.49/1.66/1.47 final pertaining validation loss and 88.2/82.5/88.1

Dev F1 score on SQuAD v1.1 [RZLL16]. We leave comprehensive evaluations for future work.

6 Conclusion

In this work, we study Transformers by constructing LEGO, a controllable synthetic logical reasoning task. With LEGO, we have gained insights into their inductive bias, the role of pertaining, etc. Based on these insights, we proposed the LEGO attention mechanism which both accelerates inference and leads to comparable or even better performance. There are many important attention heads beyond just manipulation and association, and their roles remain to be discovered. We believe LEGO will continue to deepen our understanding on Transformers’ inner working and to inspire better algorithms/architectures for tasks beyond LEGO.

References

- [AAG21] Vishesh Agarwal, Somak Aditya, and Navin Goyal. Analyzing the nuances of transformers' polynomial simplification abilities. *arXiv preprint arXiv:2104.14095*, 2021.
- [AWA⁺22] Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *arXiv preprint arXiv:2207.04901*, 2022.
- [BAG20] Satwik Bhattacharya, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, 2020.
- [BMR⁺20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.
- [CIS21] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The neural data router: Adaptive control flow in transformers improves systematic generalization. *arXiv preprint arXiv:2110.07732*, 2021.
- [CKLM19] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 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*, pages 276–286, Florence, Italy, August 2019. Association for Computational Linguistics.
- [CLR⁺21] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.
- [CND⁺22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- [CNM19] Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse transformers. In *EMNLP*, 2019.
- [CST21] Xinyun Chen, Dawn Song, and Yuandong Tian. Latent execution for neural program synthesis beyond domain-specific languages. *Advances in Neural Information Processing Systems*, 34, 2021.
- [DBK⁺21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.
- [DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [DGV⁺18] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

- [DHD⁺21] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*, 2021.
- [GSL⁺18] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. Annotation artifacts in natural language inference data. *arXiv preprint arXiv:1803.02324*, 2018.
- [HBK⁺21] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset, 2021.
- [HBM⁺22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [HDMB20] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.
- [HSL⁺16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661, 2016.
- [HSW⁺22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [JEP⁺21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.
- [KSS⁺20] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.
- [LB18] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR, 2018.
- [LBD⁺89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
- [LCG⁺19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.
- [LYWP21] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C. Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations*, 2021.

- [MPL19] R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*, 2019.
- [NAGA⁺22] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. In *Deep Learning for Code Workshop*, 2022.
- [NNSN21] Thao Nguyen, Vaishnav Nagarajan, Hanie Sedghi, and Behnam Neyshabur. Avoiding spurious correlations: Bridging theory and practice. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021.
- [NYM⁺20] Weili Nie, Zhiding Yu, Lei Mao, Ankit B Patel, Yuke Zhu, and Anima Anandkumar. Bongard-logo: A new benchmark for human-level concept learning and reasoning. *Advances in Neural Information Processing Systems*, 33:16468–16480, 2020.
- [PBBG22] Arkil Patel, Satwik Bhattacharya, Phil Blunsom, and Navin Goyal. Revisiting the compositional generalization abilities of neural sequence models. *arXiv preprint arXiv:2203.07402*, 2022.
- [PHZ⁺22] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.
- [RBC⁺21] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.
- [RDN⁺22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.
- [RKR20] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.
- [RMS⁺21] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021.
- [RST20] Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*, 2020.
- [RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.
- [RZP⁺22] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.
- [SGSB20a] Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. Prover: Proof generation for interpretable reasoning over rules. *arXiv preprint arXiv:2010.02830*, 2020.

- [SGSB20b] Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. Prover: Proof generation for interpretable reasoning over rules. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 122–136. Association for Computational Linguistics, 2020.
- [SHL20] Megha Srivastava, Tatsunori Hashimoto, and Percy Liang. Robustness to spurious correlations via human annotations. In *International Conference on Machine Learning*, pages 9109–9119. PMLR, 2020.
- [SPN⁺22] Shaden Smith, Mostafa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.
- [TDFH⁺22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [VTM⁺19] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.
- [WDS⁺20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierrick Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.
- [WRL⁺21] Yuhuai Wu, Markus N Rabe, Wenda Li, Jimmy Ba, Roger B Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. In *International Conference on Machine Learning*, pages 11251–11262. PMLR, 2021.
- [WWS⁺22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.
- [YPPN21] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*, 2021.
- [YSI20] Weiqiu You, Simeng Sun, and Mohit Iyyer. Hard-coded gaussian attention for neural machine translation. *arXiv preprint arXiv:2005.00742*, 2020.
- [ZRG⁺22] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.
- [ZRKB21] Chiyuan Zhang, Maithra Raghu, Jon Kleinberg, and Samy Bengio. Pointer value retrieval: A new benchmark for understanding the limits of neural network generalization. *arXiv preprint arXiv:2107.12580*, 2021.

A Shortcut solutions and their effect on generalization

As explained in Section 4.3, we have observed that the randomly initialized models first learn a “shortcut” solution—predicting the last variable in the chain by counting the overall number of minus signs—instead of the “common sense” iterative solution. This can be seen in the top two plots in Figure 3, where the accuracy of variable #11 improves earlier than most of the other variables.¹

To be more precise, let us describe the two solutions in detail with an example. Consider the input: $a=+1; d=-c; b=-a; c=b;$ Initially, only the variable a is resolved. The iterative solution identifies an unresolved variable that appears in the same clause with an already resolved variable, resolves it according to that clause, and repeats. In this example, it would resolve b to -1 by the clause $b=-a$, then resolve c to -1 by the clause $c=b$, and then resolve d to 1 by the clause $d=-c$. The shortcut solution identifies an unresolved variable that appears only once, and resolves it to 1 if the overall number of minus signs is even, and to -1 otherwise. In the above example, where d is the last variable in the reasoning chain, the shortcut solution correctly resolves it to 1 .

As further mentioned in Section 4.3, the shortcut solution to LEGO may be related to the phenomenon of spurious features, where models learn to perform tasks in ways that circumvents the intended “common sense” solution a human would use. Such spurious solutions are often considered undesirable, as they are known to generalize poorly even to mild variants of the task. Indeed, the shortcut solution to LEGO is brittle even under simple variations to the problem:

- Repeated clauses, e.g., $a=+1; b=-a; b=-a;$
- Redundant clauses, e.g., $a=+1; b=a; c=-a; c=-b;$
- Multiple jointly rooted reasoning chains, e.g., $a=+1; b=-a; c=-a;$
- Multiple disjoint reasoning chains, e.g., $a=+1; b=+1; c=-a; d=-b;$

and more. In all those settings the shortcut solutions would fail, whereas the “common sense” iterative solution would succeed. This motivates us to empirically study the effect of the shortcut solution on the ability of the models to generalize. We pose the following questions:

¹This behavior is not expected in the bottom two plots, since in the top ones the models are trained to fit all 12 variables including #11, while in the bottom plots the models are only trained to fit the first 6 variables, which precludes learning the shortcut solution.

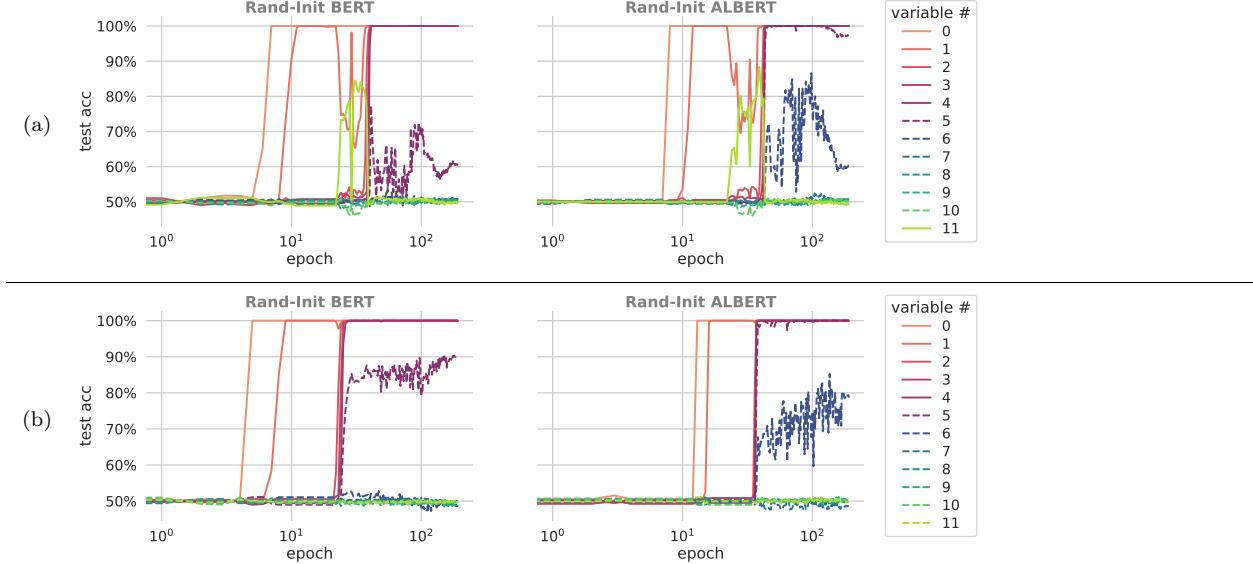


Figure 11: Learning shortcut impedes generalization. (a) Train on variables #0-#4 and #11. (b) Train on variables #0-#4 only. In both plots we test on all 12 variables.

Q1. How does reliance on shortcut solutions affect the ability of the network to generalize?

A first indication that the shortcut solution is undesirable for LEGO can be gleaned already from Figure 3: along with the early improvement in the accuracy of variable #11 (which indicates that the shortcut solution is being learned), we observe a drop in the accuracy of some of the variables that were already learned (#2 in ALBERT and #3 in BERT). This may suggest that the shortcut solution impedes even classical generalization. Indeed, the models seem to “realize” that, as we can see that the accuracy of #11 drops before improving again together with the other variables, indicating that the shortcut solution is being abandoned in favor of a solution that can predict all variables correctly.

To gain insight into the effect of the shortcut solution on out-of-distribution generalization, we performed an experiment where the models are trained to fit the first five variables (#0-#4) and the last one (#11), and are asked to predict all 12 variables at test time. This is different from the top plots in Figure 3 where the model was trained to fit all 12 variables (and thus no out-of-distribution generalization is observed), and from the bottom plots in Figure 3 where the model is trained to fit the first six variables (#0-#5), without #11 (and thus learning the shortcut solution is not possible). The results of this experiment are reported in the top two plots in Figure 11. The bottom plots depict a control experiment where the models are trained to fit only the first five variables, without #11 (and thus, again, learning the shortcut solution is not possible). The results show that the models exhibit inferior out-of-distribution generalization (to variables #5 in BERT and #6 in ALBERT) when provided supervision for #11 (top plots), even though they are given strictly more information during training than in the bottom plots, ostensibly making the task easier. We thus infer that the shortcut solution in LEGO has an adverse effect on out-of-distribution generalization.

Q2. What are effective ways to prevent models from learning shortcuts, and do they result in better generalization?

In Section 4 we studied the effect of pretraining on LEGO, and observed that the pretrained BERT model exhibits much better out-of-distribution generalization than the randomly initialized BERT model. This naturally suggests that pretrained BERT possibly avoids the shortcut solution. We confirm experimentally in Figure 12 (top), where pretrained BERT and ALBERT are finetuned to fit all 12 variables. Indeed observe

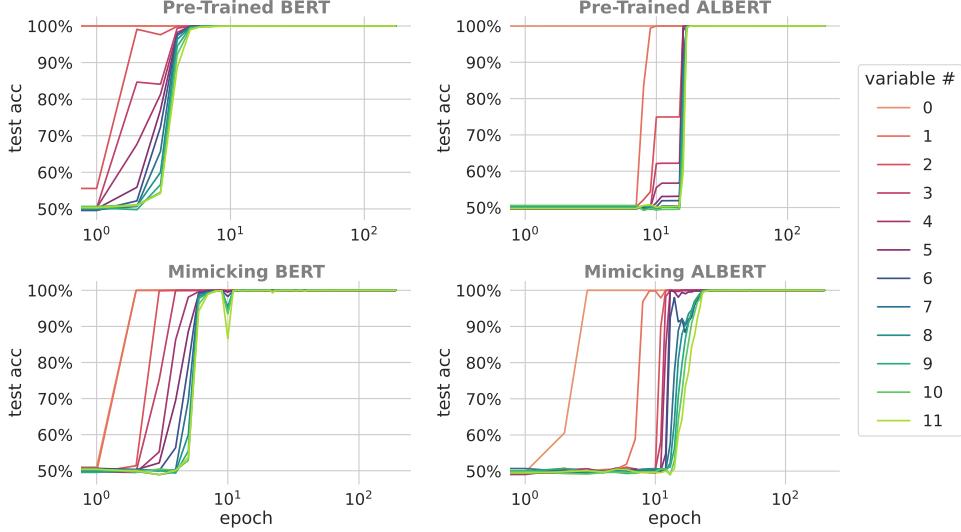


Figure 12: Shortcut solutions are avoided via either pre-training or mimicking.

the accuracy of #11 improving either later than or concurrently with all other variables, suggested that the shortcut solution is not being learned. We speculate that this may have to do with the number of epochs it takes to learn the iterative solution: By the time the randomly initialized BERT has learned the shortcut solution, pretrained BERT already attains full accuracy on the entire chain of variables. Avoiding the shortcut solution may partly explain the superior out-of-distribution generalization performance of pretrained BERT over randomly initialized BERT, seen in Figure 6.

Section 4 also showed that our mimicking technique—which directly mimics the attention patterns of the pretrained model without training on any data—can recover much of the benefit in pretraining for LEGO. This extends to avoiding the shortcut solution as well: the bottom plots in Figure 12 show that the mimicking BERT and ALBERT models exhibit similar accuracy patterns to their pretrained counterparts, suggesting that the shortcut solution is not being learned by them.

B Data generation and training details for the LEGO task

B.1 Data generation

We specify the data generation mechanism for Task 1 in the following. We use lowercase alphabets as variables $A = \{a, b, c, d, \dots, z\}$. Given n , we generate a sentence from our distribution $s \sim \mathcal{D}(n)$ as follows:

1. Sample n variables $a_1, a_2, \dots, a_n \in A$ and their corresponding assignments (or labels) $y_1, y_2, \dots, y_n \in X$ uniformly, i.e., $\forall i \in [n], a_i \sim \text{Unif}(A)$ and $y_i = \pm 1$ w.p. 0.5.
2. The n clauses are then generated as $a_i = g_i a_{i-1}$ for $i = 1, 2, \dots, n$, where $a_0 = r = 1$ and the group elements $g_1, g_2, \dots, g_n \in G$ are uniquely chosen so that the clauses are consistent with assignments $a_i = y_i$ for all $i \in [n]$.
3. The sentence s is generated by concatenating a random ordering of the n clauses with a semicolon ;. Finally, the sentence is padded with [BOS] and [EOS] tags to denote the beginning and end of sentence, respectively. See Figure 13 for example sentences from our distribution.

```
[BOS] j=-f; f=-b; y=+t; o=+e; d=+y; v=+d; h=-o; b=-i; i=+1; t=+l; e=-j; l=-h; [EOS]
[BOS] j=+o; s=-y; p=-r; y=-m; u=-a; a=-f; k=+p; o=-k; q=+u; m=+1; f=+s; r=+q; [EOS]
[BOS] z=+d; b=+1; m=+t; d=-u; u=-h; a=-b; j=+m; i=-j; t=+x; f=+i; h=-f; x=-a; [EOS]
[BOS] j=-f; f=-b; y=+t; o=+e; d=+y; v=+d; h=-o; b=-i; i=+1; t=+l; e=-j; l=-h; [EOS]
[BOS] w=+l; m=+c; c=-i; f=-d; p=-m; a=+b; y=-a; b=+p; i=+f; l=-v; d=+1; v=+y; [EOS]
```

Figure 13: Samples of sentences generated from our distribution $\mathcal{D}(n)$ for Task 1 with $n = 12$.

B.2 Training

Our vocabulary for data generated as above thus consists of symbols of variables $a \in A$, group operations $+, - \in G$, the root node $1 \in X$, the equal sign ‘=’, and the semicolon ‘;’ along with the [BOS] and [EOS] tags. To apply transformers to the LEGO task we convert each symbol in our vocabulary to vectors in \mathbb{R}^d (referred to as *tokens*) using a learnable linear embedding layer. Thus, a sentence of n clauses now corresponds to an ordered list of $5n + 2$ tokens, which we turn into an unordered list using positional embedding (see [VSP⁺17]). These tokens are processed iteratively by *transformer blocks*. Each transformer block maps $5n$ tokens in \mathbb{R}^d to another set of $5n$ tokens using a multi-head attention layer followed by a one-hidden layer feedforward net (there are also residual connections and layer normalization, for full details of architecture see [VSP⁺17]). We use `bert-base-uncased`² and `albert-base-v1`³ along with their pretrained weights from the open source Huggingface transformers library [WDS⁺20]. The Rand-Init models have identical configurations to their pretrained counterparts but randomly initialized weights.

Our training and test datasets are i.i.d. samples from our distribution $D(n)$ as described above. We generate $10^4 \times n$ and $10^3 \times n$ datapoints for training and test, respectively, and sanity checked that there is no overlap between train and test data. Recall that during training we provide supervision on the first n_{tr} appearing in the graph representation of the sentence, but test the accuracy on all n samples at test time. Note that since the clause positions are randomized in our input sentence (e.g., Figure 13), the first n_{tr} clauses in the graph representation can appear at arbitrary positions in the sentence which allows for training the positional encodings for longer sequences than those seen in training.

In all the LEGO experiments, we use cross entropy loss averaged over the n_{tr} clauses as our sample loss during training. We train for 200 epochs using the Adam optimizer with batch size of 1000 samples, 5×10^{-5} learning rate, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and cosine learning rate schedule with $T_{\max} = 200$. For tokenization, we use the pre-trained BERT tokenizer for all the experiments, which merely converts the input symbols into integers that are used as token ids. Each run is conducted on a cluster with 4 A100 GPUs.

A note on variance of training across LEGO tasks When training BERT and ALBERT models for the LEGO task using our experimental setup, we see non-trivial variance in absolute test accuracies across different runs. In Figure 14, we show three different runs of BERT and ALBERT models trained on LEGO tasks of length $n = 12$ (the configuration used in most results in the paper). While we see that the absolute values of the test accuracies vary significantly, the qualitative observations made in our paper hold across all the runs: importantly, across all the runs, we see that using iterative ALBERT architecture as well as pre-training non-iterative BERT architecture leads to better generalization to unseen lengths. This shows that conclusions derived in our paper hold despite the variance across runs.

Methodologically, we attribute the variance in the absolute test accuracy to the relatively small size of our datasets (e.g., the number of training examples for $n = 12$ is 120K tokens) for training standard trained language models which are otherwise trained on hundreds of millions of tokens. Furthermore, note that in our experiments the variance across runs arise both from having different train and test datasets as well as different random seeds for model initialization and training algorithm.

²<https://huggingface.co/bert-base-uncased>

³<https://huggingface.co/albert-base-v1>

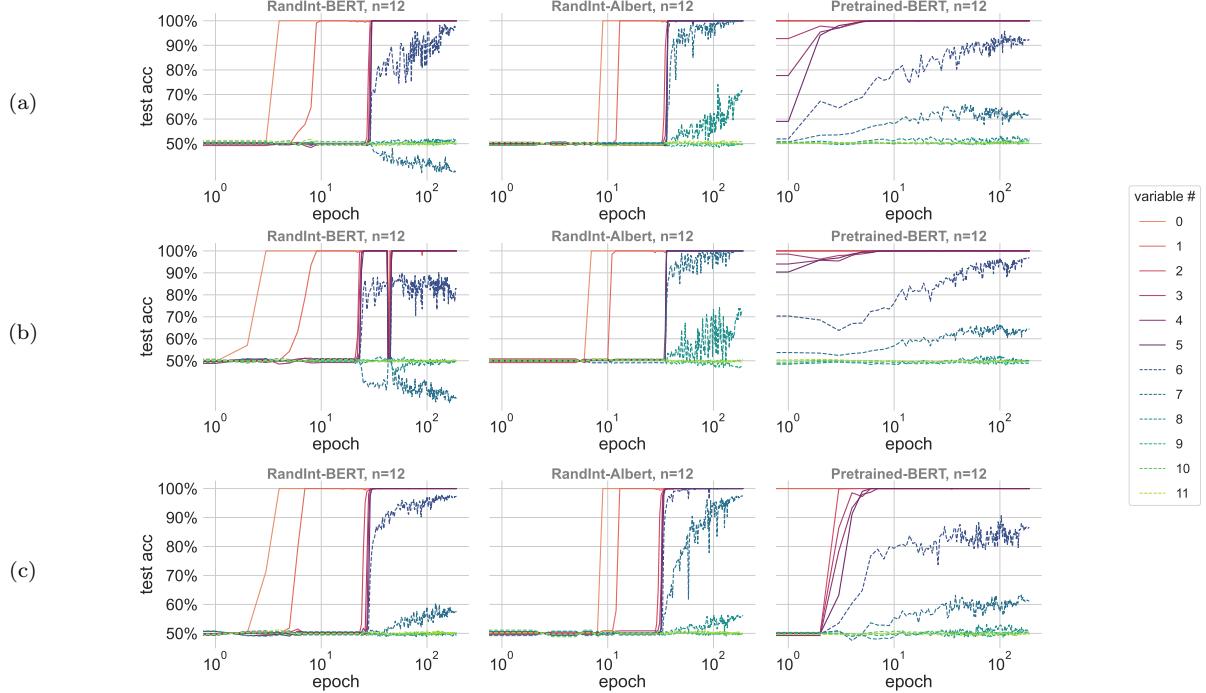


Figure 14: Three sample runs of models trained on LEGO tasks with $n = 12$ and $n_{tr} = 6$. We observe that while there are variances across different runs of the models, the qualitative conclusions stated in the paper holds for all the runs: i.e, iterative ALBERT models and pretraining lead to better generalization to unseen task lengths.

B.3 The Mimicking procedure

Starting with a randomly initialized transformer model, we would like to make sure that some of its attention heads implement the manipulation and association functionalities, *prior to* the fine-tuning. To do so, we craft the desired attention patterns of a manipulation head and an association head, and train the model with gradient descent to match them. Specifically, given a random input $x \in \mathbb{R}^T$ of length T , we hard code the following matrix $M \in \mathbb{R}^{T \times T}$ as the target attention pattern for the manipulation head⁴, and derive from input x the following matrix $A \in \mathbb{R}^{T \times T}$ whose $A_{i,j}$ entry indicates whether x_i and x_j are identical tokens. Note that we further specify that A has a zero diagonal in observance of the association head in Figure 7 having a vanishing diagonal. In reality, attention maps have unit row sums, thus we normalize M and A accordingly to obtain \tilde{M} and \tilde{A} such that their rows $\tilde{M}[t, :]$ and $\tilde{A}[t, :]$ are valid distributions.

$$M = \begin{bmatrix} \ddots & \ddots & \ddots \\ 1 & 2 & 4 & 2 & 1 \\ & 1 & 2 & 4 & 2 & 1 \\ & & 1 & 2 & 4 & 2 & 1 \\ & & & \ddots & \ddots & \ddots \end{bmatrix}, \quad A_{ij} = \begin{cases} 1, & \text{if } x_i = x_j \text{ and } i \neq j \\ 0, & \text{otherwise} \end{cases}$$

At every layer, we randomly appoint two attention heads to mimic the manipulation and association operators while leave the other heads off the hook. Upon seeing a input sequence $\mathbf{x} \in \mathbb{R}^T$, we denote the

⁴Here we choose the Gaussian filter [1, 2, 4, 2, 1] for illustration. In practice, we find that the final performance is robust to various pattern choices as long as they are localized and shift-invariant.

attention maps of the appointed heads at the l -the layer as $\text{Attn}_0^{(l)}(\mathbf{x}), \text{Attn}_1^{(l)}(\mathbf{x}) \in \mathbb{R}^{T \times T}$. For the mimicking objective, we draw input sequence \mathbf{x} 's whose tokens are independent and uniform over the vocabulary, and then compute the Kullback–Leibler divergence between each row of $\text{Attn}_0^{(l)}(\mathbf{x}), \text{Attn}_1^{(l)}(\mathbf{x})$ and the corresponding rows of \tilde{M}, \tilde{A} . Thus the overall mimicking loss is

$$L_{\text{mimic}} = \underset{\text{rand. seq. } \mathbf{x}}{\mathbb{E}} \left[\sum_{l=0}^{L-1} \frac{1}{T} \sum_{t=1}^T \left[\text{KL} \left(\text{Attn}_0^{(l)}(\mathbf{x})[t, :] \middle\| \tilde{M}[t, :] \right) + \text{KL} \left(\text{Attn}_1^{(l)}(\mathbf{x})[t, :] \middle\| \tilde{A}[t, :] \right) \right] \right]$$

Note that the above mimicking loss pertains only two attention heads per layer and we leave out all the rest. Then the mimicking procedure boils down to updating the transformer model's parameters to minimize the L_{mimic} . We find that a vanilla Adam optimizer drives the mimicking loss down to near zero in a negligible amount of time compared to large-scale pre-training, even for large models such as BERT and ALBERT.

C Details of the LEGO v0/v1 models

The LEGO v0/v1 models are derived from the BERT-base model. They have same number of layers, same hidden dimensions, and same feed-forward layer's structure as the BERT model while they differ in attention layers. In all three models, each attention head effectively operates on 64 hidden dimensions. The specifics are provided below.

- BERT-base model has 12 layers, 768 hidden dimensions, and 12 attention heads per layer.
- LEGO v0 model: convolutional pathway consists of Linear from 768 dim to 576 dim + ReLU + Depthwise temporal Convolution with kernel size 21 and 576 channels + Linear from 576 dim to 576 dim. There are three hardcoded attention heads whose attention patterns are $A_{asso}, A_{cls}, A_{sep}$ and each of them operates on a value map of 64 dim (thus the entire value map is a Linear layer from 768 dim to 192 dim).
- LEGO v1 model: convolutional pathway consists of Linear from 768 dim to 384 dim + ReLU + Depthwise temporal Convolution with kernel size 21 and 384 channels + Linear from 384 dim to 384 dim. There are three hardcoded attention heads whose attention patterns are $A_{asso}, A_{cls}, A_{sep}$ and each of them operates on a value map of 64 dim (thus the entire value map is a Linear layer from 768 dim to 384 dim). There are also 3 ordinary attention heads whose Q,K,V maps are all Linear layers from 768 dim to 384 dim.

D Effect of length of LEGO chains and depth of model

All our experiments in the main paper were on a typical instance of LEGO task with length $n = 12$ chains and standard BERT and ALBERT models of depth $D = 12$. In this Appendix, we briefly explore the effect of the LEGO chain length (n, n_{tr}) and transformer models depth D . The chain structure of information flow in our LEGO Task 1 would suggest that for a transformer network of depth D , the learning and generalization on LEGO task would crucially depend on its maximum chain length n . If n (or importantly n_{tr}) is too small, the training data might not have enough information to guide generalization to longer lengths, on the other hand, if n is too large, the model might not be able to propagate information along the chain in a natural iterative manner. For example, implementing a “natural” iterative algorithm of resolving one clause of the task at a time (as described in the beginning of Appendix A) would require models of depth $D \geq n$.

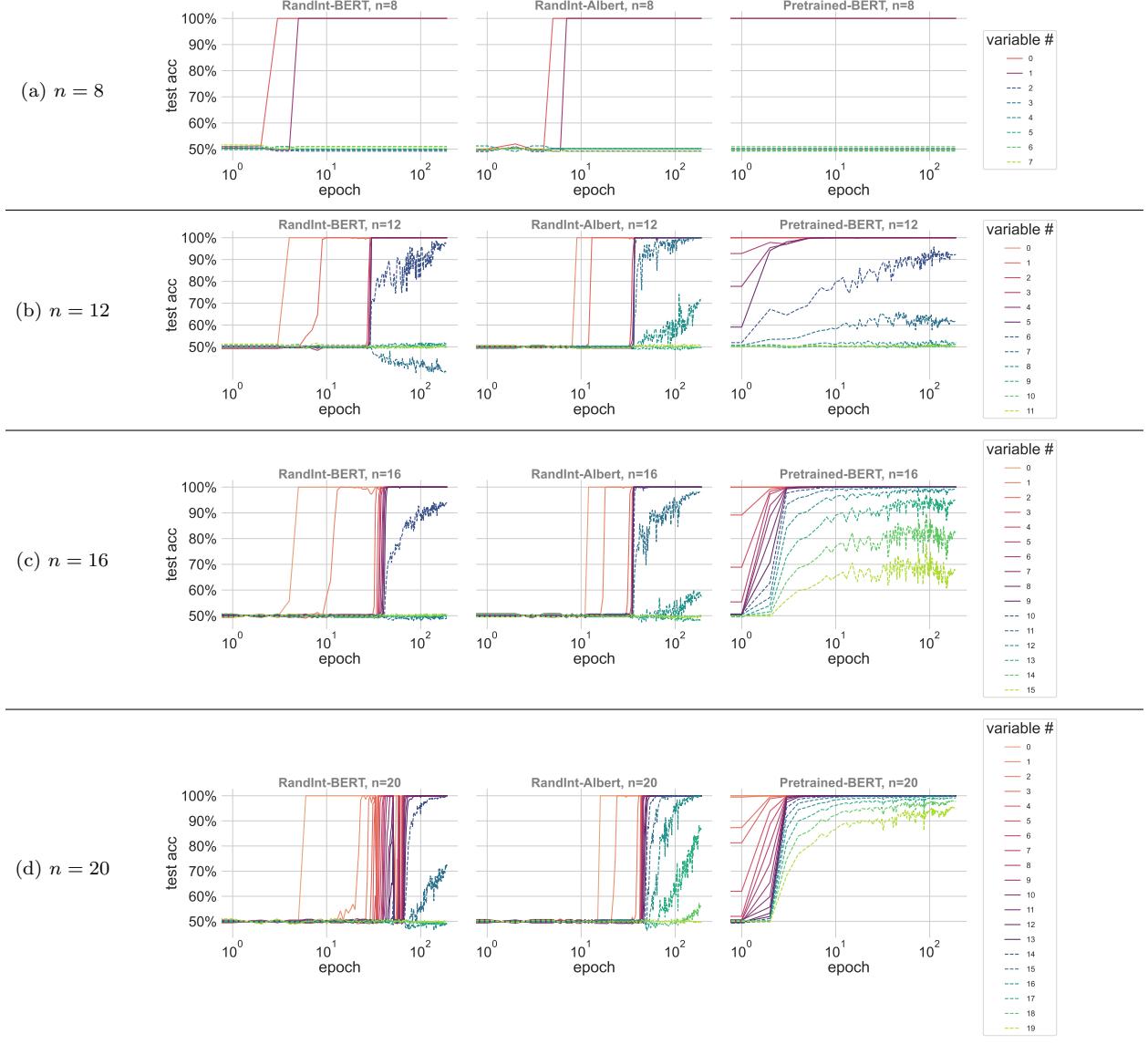


Figure 15: Generalization performance of BERT and ALBERT models (depth $D = 12$) trained on LEGO task of varying chain lengths n : (a) $n = 8$, (b) $n = 12$, (c) $n = 16$, (d) $n = 20$.

We study this behavior by first repeating the experiments from our main paper on depth $D = 12$ models on LEGO tasks of varying lengths n . In all the cases, we proportionally increase the length of chain for which supervision is provided by training on first $n_{tr} = n - 6$ clauses in the chain.

In Figure 15, we show the performance of a typical run of transformer models in this setting. When trained on small length chains of $n = 8$ and $n_{tr} = 2$, we indeed see that all the models including pretrained-BERT is able to learn the short training length (classical generalization) but the supervision does not contain enough information for the models to learn to generalize to unseen lengths. On the other hand, for larger chain lengths we see that the generalization only gets better, with a very strong monotonic trend for pretrained-BERT models. In particular, the strong generalization performance at $n = 20$ with $n_{tr} = 14$ would suggest that these models might not really be implementing the “natural” iterative solution for the task. Rather, it is possible that training on longer sequences leads the models to learn a more compact representation that

nevertheless generalizes remarkably well to much longer lengths than seen during training.

Complementing our results on varying the length of LEGO chains, we also look at how BERT and ALBERT architectures of smaller depth $D < 12$ learn our LEGO task of length $n = 12$. In Figure 16 we show the results of models trained from random initialization (we do not have pre-trained models at these depths). Here we do see the trend that larger depth improves generalization. In particular, for a task of chain length n , there does appear to be a minimum threshold of D at which the models learn to generalize even in the classical sense (on lengths the models were trained on), but this relationship appears sub-linear rather than linear with $D \geq n$ as speculated by the “natural” iterative algorithm. It would be of interest in future studies to explore this relation between depth and length better.

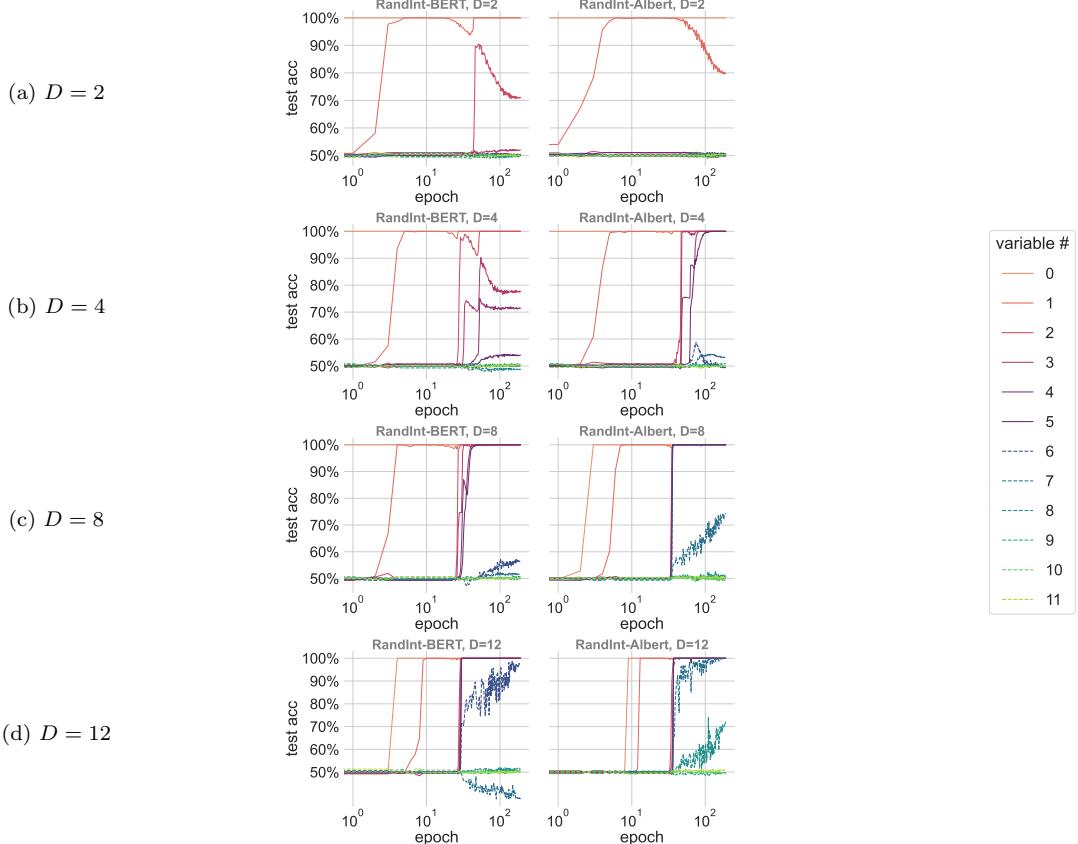


Figure 16: Rand-Init BERT and ALBERT models of varying depth D trained on LEGO tasks of length $n = 12$.

D.1 Effect of number of parameters

In this section, we provide empirical evidence that the advantage of ALBERT over BERT for predicting longer sequences does not merely come from fewer parameters. Precisely, we modify the original BERT model to have 96 hidden size instead of 784 uniformly across all layers, with all other hyper-parameter unchanged. This “thin” BERT model has approximately the same parameter count as the ALBERT model. The result is shown in Figure 17.

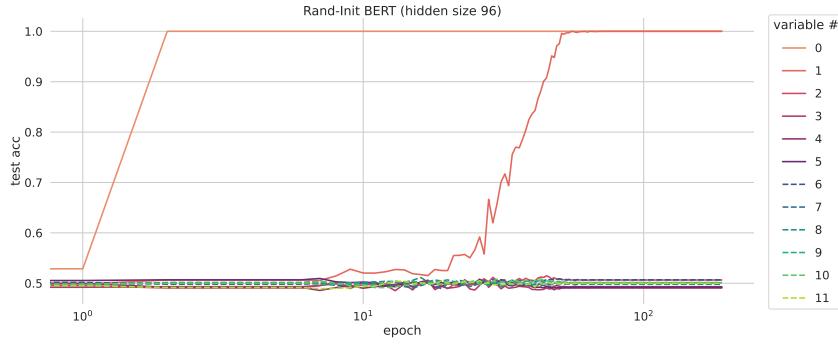


Figure 17: The "thin" model not only lacks the length extrapolation properties of ALBERT, but it does not even generalize in-distribution. In fact, test accuracy on variable #2 remains near 50% after 200 training epochs.

E LEGO Task 2: dihedral group

As a generalization of the main task (i.e., LEGO Task 1) analyzed so far, we present LEGO Task 2 of learning the dihedral group⁵ D_3 of order 6 which is isomorphic to the symmetric group S_3 . Note that LEGO Task 1 can be viewed as learning the dihedral group D_1 of order 2. Clearly, the shortcut solution described in Section A is not valid here.

We repeat the out-of-distribution generalization experiments on Task 2 with the exact same model configurations and training hyper-parameters as Task 1 (see Section B.2). For dataset creation, we largely follow the pipeline detailed in Section B.1 with group elements from D_3 for which we create corresponding tokens. The only modification here is that the labels are categorical with 6 classes, since every variable in Task 2 may take 6 candidate values.

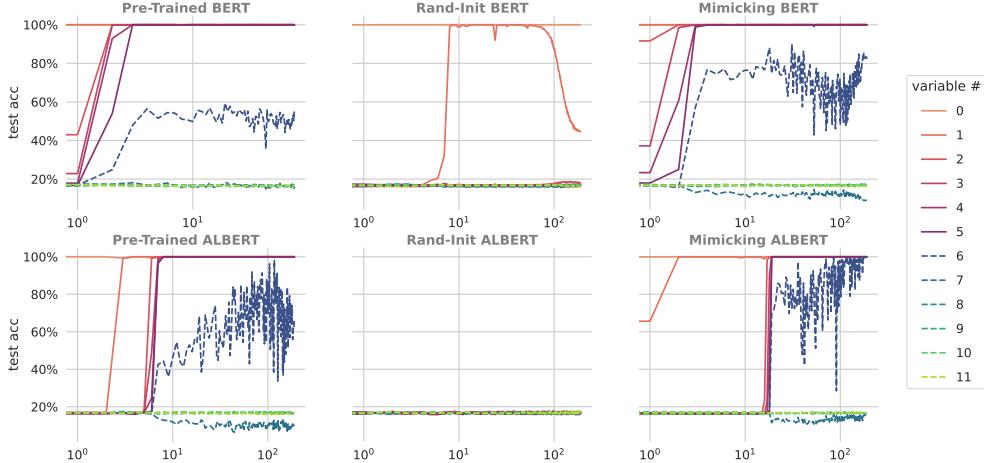


Figure 18: Out-of-distribution generalization on LEGO Task 2. Task 2 appears to be significantly more challenging than Task 1 and pretraining plays a more important role on top of rand-init models, while the mimicking procedure is able to match and even outperform pretraining.

In Figure 18, we show these preliminary results and observe that Task 2 is indeed more challenging

⁵https://en.wikipedia.org/wiki/Dihedral_group

than Task 1, and the extent to which pretraining provides benefits is noticeably larger. Without further hyper-parameter tuning, the randomly initialized models even face optimization issues in fitting the training labels. Interestingly, we also find our proposed mimicking procedure introduced in Section B.3 is not only able to match pretraining’s performance and also outperform it.

So far, none of the models here is capable of non-trivially generalizing to more than one extra variable. It is an important future direction to search for suitable adaptations to the current transformer architectures/training algorithms that can eventually solve this task.

Furthermore, the amount of knowledge that models have to ”memorize”, e.g. the outcomes of $g(x)$ for all $g \in G$ and $x \in X$ grows quadratically with group size. While LEGO task 1 contains only 4 mappings, namely $1 = +1, -1 = +(-1), 1 = -(-1), -1 = +(-1)$. For moderately size group such as D_5 , models will have to ”memorize” $(5!)^2 = 125^2$ in total mappings. This makes the task perfectly suitable for analyzing how and where (at which neurons) the models store these informations. We consider this as an exciting future direction.

F Attention maps of pretrained BERT on LEGO

To complement the visualizations in Figure 7, we provide visualizations of attentions maps of all attention heads from a pretrained BERT when taking a LEGO sequence as input. We make the observation that *almost every* attention head implements either a manipulation or an association operator. Please view electronically and zoom in for details. All the figures are vectorized.

