Making Transformers Solve Compositional Tasks

Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, Zachary Fisher
Google Research
{santiontanon, jainslie, vcvicek, zachfisher}@google.com

Abstract

Several studies have reported the inability of Transformer models to generalize compositionally, a key type of generalization in many NLP tasks such as semantic parsing. In this paper we explore the design space of Transformer models showing that the inductive biases given to the model by several design decisions significantly impact compositional generalization. We identified Transformer configurations that generalize compositionally significantly better than previously reported in the literature in many compositional tasks. We achieve state-of-the-art results in a semantic parsing compositional generalization benchmark (COGS), and a string edit operation composition benchmark (PCFG).

1 Introduction

Although modern neural network architectures reach state-of-the-art performance in many challenging natural language tasks, they seem to exhibit a low amount of "compositional generalization", i.e., the ability to learn a set of basic primitives and combine them in more complex ways than those seen during training (Hupkes et al., 2020). For example, suppose a system has learned the meaning of "jump" and that "jump twice" means that the action "jump" has to be repeated two times. Upon learning the meaning of the action "jax", it should be able to infer what "jax twice" means. Compositional generalization is a key aspect of natural language and many other tasks we might want machine learning models to learn.

While both humans and classical AI techniques (such as grammars or search-based systems) can handle compositional tasks with relative ease, it seems that modern deep learning techniques do not possess this ability. A key question is thus: Can we build deep learning architectures that can also solve compositional tasks? In this paper we focus on Transformers (Vaswani et al., 2017), which have been shown in the literature to exhibit poor compositional generalization (see Section 2). Through an empirical study, we show that this can be improved. With the goal of creating general models that generalize compositionally in a large range of tasks, we show that several design decisions, such as position encodings, decoder type, weight sharing, model hyper-parameters, and formulation of the target task result in different inductive biases, with significant impact for compositional generalization[1]. We use a collection of twelve datasets designed to measure compositional generalization. In addition to six standard datasets commonly used in the literature (such as SCAN (Lake and Baroni, 2018), PCFG (Hupkes et al., 2020), CFQ (Keysers et al., 2019) and COGS (Kim and Linzen, 2020)), we also use a set of basic algorithmic tasks (such as addition, duplication, or set intersection) that although not directly involving natural language, are useful to obtain insights into what can and cannot be learned with different Transformer models. We also include tasks where we do not see significant improvements, to understand what types of compositional generalization are improved with our proposed modifications, and which are not.

The main contributions of this paper are: (1) A study of the Transformer design space, showing which design choices result in compositional learning biases across a variety of tasks. (2) state-of-the-art results in COGS, where we report a classification accuracy of 0.784 using an intermediate representation based on sequence tagging (compared to 0.35 for the best previously reported model (Kim and Linzen, 2020)), and the productivity and systematicity splits of PCFG (Hupkes et al., 2020).

The rest of this paper is organized as follows. Section 2 provides some background on compositional generalization and Transformers. In Sec-

[1]Source code: https://github.com/google-research/google-research/tree/master/compositional_transformers.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 3591 - 3607

tion 3, we present the datasets used in our empirical evaluation, which is presented in Section 4. The paper closes with a discussion on the implications of our results, and directions for future work.

2 Background

This section briefly provides background on compositional generalization and Transformer models.

2.1 Compositional Generalization

Compositional generalization can manifest in different ways. Hupkes et al. (2020) identified five different types, such as *systematicity* and *productivity* (extrapolation to longer sequences than those seen during training). *Systematicity* is the ability of recombining known parts and rules in different ways than seen during training. The example in the introduction of knowing the meaning of "jump", "jump twice" and "jax" and from those inferring the meaning of "jax twice" is an example of systematicity. *Productivity*, on the other hand, is the ability to extrapolate to longer sequences than those seen during training. For example, consider the example of learning how to evaluate mathematical expressions of the form "$3 + (4 - (5 * 2))$". An example of productivity would be to extrapolate to expressions with a larger number of parenthesis, or with deeper parenthesis nesting, than seen during training. Hupkes et al. (2020) identify other forms of compositionality, such as *substitutivity*, *localism* or *overgeneralization*, but we will mostly focus on systematicity and productivity in this paper.

Compositional generalization is related to the general problem of *out-of-distribution generalization*. Hence, we can also see it as the problem of how models can discover *symmetries* in the domain (such as the existence of primitive operations or other regularities) that would generalize better to out-of-distribution samples than *shortcuts* (Geirhos et al., 2020), which would only work on the same distribution of examples seen during training.

Early work focused on showing how different deep learning models do not generalize compositionally (Liška et al., 2018). For example Liška et al. (2018) showed that while models like LSTMs are able to generalize compositionally, it is unlikely that gradient descent converges to a solution that does so (only about 2% out of 50000 training runs achieved a generalization accuracy higher than 80% in a compositional task, while they had almost perfect performance in training). Datasets like SCAN (Lake and Baroni, 2018), PCFG (Hupkes et al., 2020), Arithmetic language (Veldhoen et al., 2016), or CFQ (Keysers et al., 2019) were proposed to show these effects.

Work toward improving compositional generalization includes ideas like Syntactic attention (Russin et al., 2019), increased pretraining (Furrer et al., 2020), data augmentation (Andreas, 2019), intermediate representations (Herzig et al., 2021) or structure annotations (Kim et al., 2021). Specialized architectures that achieve good performance in specific compositional generalization tasks also exist. For example, Liu et al. (2020) propose a model made up of a "composer" and a "solver", achieving perfect performance on SCAN. The most related concurrent work to ours is that of Csordás et al. (2021), who also showed gains in compositional generalization via relative attention. Additionally, in their work, they show that a key problem in some tasks is the *end of sequence* detection problem (when to stop producing output). Finally, they show that generalization accuracy keeps growing even when training accuracy maxes out, questioning early stopping approaches in compositional generalization. We note that training for longer might also improve our results, which we will explore in the future.

2.2 Transformer Models

Models based on Transformers (Vaswani et al., 2017), such as BERT (Devlin et al., 2018), or variants (Yang et al., 2019; Lan et al., 2019; Raffel et al., 2019) yield state-of-the-art results in many NLP tasks such as language modeling (Child et al., 2019; Sukhbaatar et al., 2019; Rae et al., 2019; Kitaev et al., 2020), question answering (Ainslie et al., 2020; Lan et al., 2019; Zaheer et al., 2020; Beltagy et al., 2020), and summarization (Zhang et al., 2019). However, existing studies show that they do not have good compositional generalization. In this paper we will consider the original Transformer architecture and expand upon it.

The standard Transformer model consists of two main components (see the center of Figure 2): an *encoder* and a *decoder*, each of which consists of a series of layers. Each layer contains an attention sublayer followed by a feed-forward sublayer (the decoder has two attention sublayers for decoder-to-decoder and decoder-to-encoder attention). The input of a Transformer is a sequence of token embeddings, and the output is a sequence of tokens

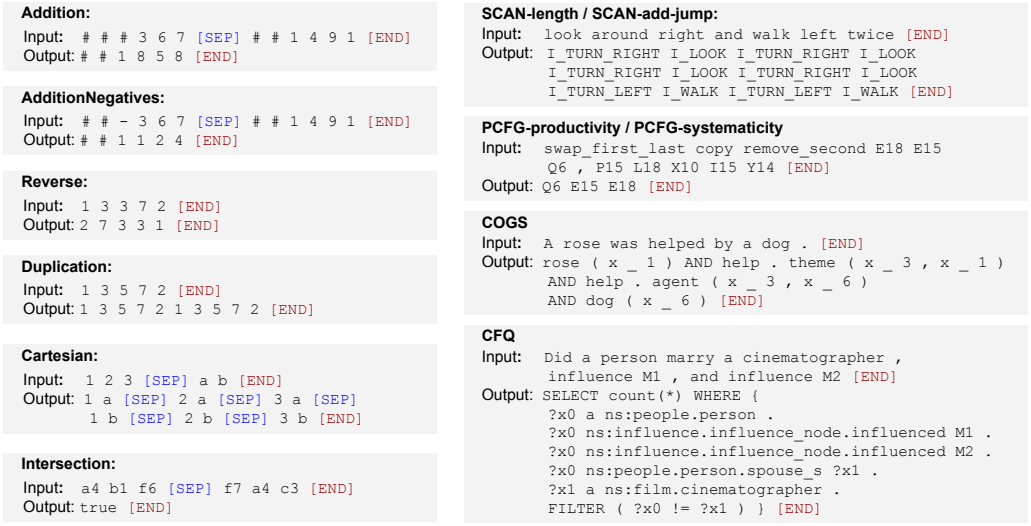

Figure 1: Examples from the different datasets used in our experiments.

generated one at a time by predicting based on the output distribution generated by the decoder. To provide a notion of token "order" a set of *position encodings* are typically added to the embedding of each input token to indicate sequence order.

We will use l to denote the number of encoder/decoder layers, d for the dimensionality of token embeddings, f for the intermediate dimensionality used by the feed-forward sublayer, and h for the number of *attention-heads* in the attention sublayers. The original Transformer model used $l = 6$, $d = 512$, $f = 2048$ and $h = 8$, as their *base* configuration. In this paper, we use parameters much smaller than that, as we are evaluating the architectural decisions on relatively small datasets.

3 Evaluation Datasets

We use a collection of 12 datasets that require different types of compositional generalization. Six of those dataset consist of "algorithmic" tasks (addition, reversing lists, etc.), and six of them are standard datasets used to evaluate compositional generalization (most involving natural language). We note that our algorithmic tasks mostly require *productivity*-style compositional generalization, while other datasets also require *systematicity* or *synonimity* (Hupkes et al., 2020). Specifically, we used the following datasets (see Appendix E for details, and Figure 1 for examples):

Addition (*Add*): A synthetic addition task, where the input contains the digits of two integers, and the output should be the digits of their sum. The training set contains numbers with up to 8 digits, and the test set contains numbers with 9 or 10 digits. Numbers are padded to reach a length of 12.

AdditionNegatives (*AddNeg*): The same as the previous one, but 25% of the numbers are negative (preceded with the − symbol).

Reversing (*Reverse*): Where the output is expected to be the input sequence in reverse order. Training contains sequences of up to 16 digits, and the test set contains lengths between 17 to 24.

Duplication (*Dup*): The input is a sequence of digits and the output should be the same sequence, repeated twice. Training contains sequences up to 16 digits, and test from 17 to 24.

Cartesian (*Cart*): The input contains two sequences of symbols, and the output should be their Cartesian product. Training contains sequences of up to 6 symbols (7 or 8 for testing).

Intersection (*Inters*): Given two sequences of symbols, the output should be whether they have a non-empty intersection. Training contains sets with size 1 to 16, and testing 17 to 24.

SCAN-length (*SCAN-l*): The *length split* of the SCAN dataset (Lake and Baroni, 2018).

SCAN-add-jump (*SCAN-aj*): The *add primitive jump split* of the SCAN dataset (Lake and Baroni, 2018).

PCFG-productivity (*PCFG-p*): The productivity split of the PCFG dataset (Hupkes et al., 2020)

PCFG-sytematicity (*PCFG-s*: The systematicity split of the PCFG dataset (Hupkes et al., 2020).

COGS: The generalization split of the COGS semantic parsing dataset (Kim and Linzen, 2020).

CFQ-mcd1 (*CFQ*): The MCD1 split of the CFQ dataset (Keysers et al., 2019).

Note that most of these datasets are trivial if

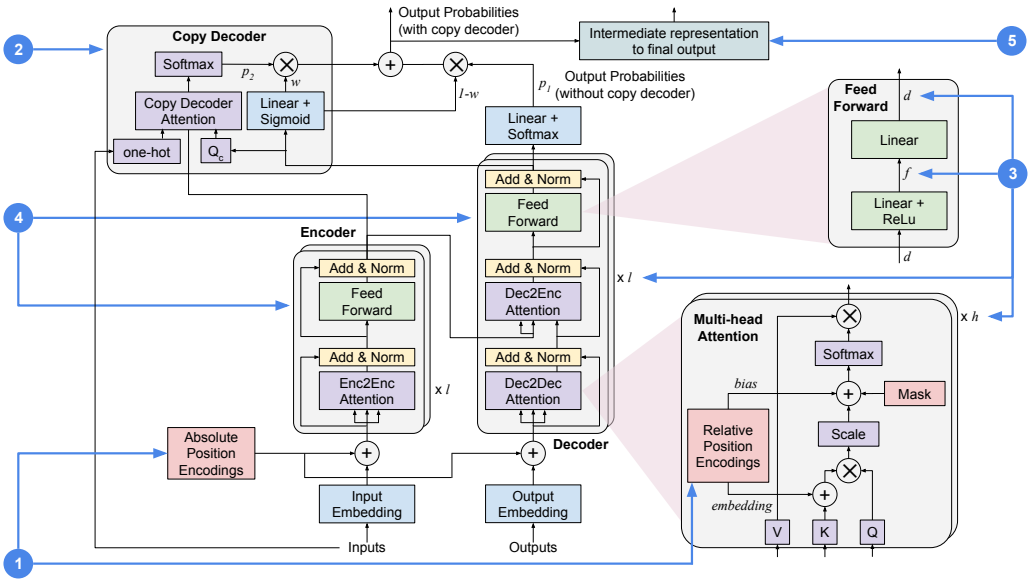

Figure 2: An illustration of a Transformer, extended with the additional components necessary to explore the different dimensions we experiment with in this paper: **(1)** position encodings, **(2)** copy decoder, **(3)** model size (l, d, f, h), **(4)** weight sharing, and **(5)** intermediate representations.

the training and test sets come from the same distribution, and most Transformer models achieve near 100% accuracy (except a few hard tasks like the Cartesian product or set intersection). Hence, splitting train and test data in a way that requires compositional generalization is key (e.g., having examples with larger sequences in the test set than in the training set). We want to make sure models do not just learn *shortcuts* (Geirhos et al., 2020) that do not generalize to out-of-distribution data.

4 Empirical Results

In this section we present an evaluation of the compositional generalization abilities of Transformers with different architectural configurations. Specifically we evaluated: (1) the type of position encodings, (2) the use of copy decoders, (3) model size, (4) weight sharing, and (5) the use of intermediate representations for prediction (see Figure 2). For this systematic experimentation, we used small Transformer models, without pre-training (all models are trained from scratch). Even if previous work has reported benefits of pre-training in some compositional tasks (e.g., in CFQ (Furrer et al., 2020)), we aim at disentangling the effects of each architecture decision in and of itself, in the search for compositional inductive biases.

Our results show that, while these decisions do not affect certain types of compositional generalization tasks, we see significant gains in others.

We report the average of at least 3 training runs (for algorithmic tasks, we use at least 5 training runs, and 10 for set intersection since they have a higher variance; see Appendix B). We use *sequence-level accuracy* as the evaluation metric: an output sequence with even just a single wrong token is considered wrong.

4.1 Position Encodings

While the original Transformer model (Vaswani et al., 2017) and BERT (Devlin et al., 2018) used *absolute position encodings*, later models such as T5 (Raffel et al., 2019) or ETC (Ainslie et al., 2020) use *relative position encodings* (Shaw et al., 2018). Relative position encodings assign a *label* to each pair of tokens in the input (typically representing their relative distance in the input, up to a maximum radius). So, there is a label used for tokens attending to a token "two positions to the right", etc. One interesting thing about relative position encodings is that they are *position invariant*, i.e. two tokens that are k positions apart will attend to each other in the same way, regardless of where they are in the sequence, and hence allowing models to capture further *symmetries* in the domain. We compare the following position encodings:

abs: sinusoidal absolute position encodings (as used in the original Transformer)[2].

[2]We did not experiment with learnable absolute position encodings, as test examples are longer than anything seen during training, hence containing untrained embeddings.

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg.
abs	0.005	0.042	0.000	0.000	0.000	0.500	0.000	0.003	0.174	0.434	0.177	0.304	0.137
rel-e	0.004	0.018	0.422	**0.486**	0.004	0.501	0.064	0.003	0.238	0.451	0.170	**0.322**	0.224
rel-b	0.002	0.005	0.277	0.362	**0.054**	0.501	0.049	0.007	0.042	0.102	0.126	0.276	0.150
rel-eb	0.003	0.011	0.486	0.444	0.000	0.500	0.089	**0.011**	**0.257**	**0.452**	0.249	0.290	0.233
rel2-e	**0.988**	**0.830**	**0.787**	0.010	0.000	0.501	0.032	0.007	0.159	0.353	**0.259**	0.322	**0.354**
rel2-b	0.140	0.708	0.056	0.253	0.000	**0.504**	0.080	0.002	0.041	0.117	0.138	0.319	0.197
rel2-eb	0.978	0.779	0.737	0.017	0.000	0.504	**0.091**	0.010	0.194	0.374	0.159	0.311	0.346

Table 1: Sequence-level accuracy for different position encoding methods. Bolded results represent the best results for each dataset in this table.

rel-e: relative position encodings, where the relative position label defines a learnable embedding that is added to the *key* during the attention process. We used a maximum local attention radius of 16, which means that we have the following relative position labels $\{l_{-16}, l_{-15}, ..., l_{-1}, l_0, l_1, ..., l_{15}, l_{16}\}$. Tokens that are further than 16 positions apart get the l_{-16} or l_{16} labels.

rel-b: relative positions define a learnable bias that is added to the attention weight of each attention pair. This is the attention mechanism used by T5 (although they use a logarithmic scheme for representing relative positions).

rel-eb: relative position using both a learnable embedding vector and a learnable bias scalar.

While relative positions are straightforward for encoder-to-encoder and decoder-to-decoder attention, it is unclear what the relative positions should be for decoder-to-encoder. Hence, we tested three alternatives (**rel2-e**, **rel2-b** and **rel2-eb** in our result tables). *rel-** methods do not use relative position labels in decoder to encoder attention, while those named *rel2-** do (where token y_i in the decoder attending to token x_j in the encoder will have label l_{j-i}.

Table 1 shows sequence-level classification accuracy for *small* Transformers ($l = 2$, $d = 64$, $f = 256$, $h = 4$). The right-most column shows the average accuracy across all datasets, and we can see that position encodings play a very significant role in the performance of the models. Going from 0.137 accuracy of the model with absolute position encodings up to 0.354 for a model with relative position encodings using embeddings (but no bias term), as well as relative positions for decoder-to-encoder attention. In general almost any type of relative position encodings help, but using embeddings helps more than using bias terms. Moreover, position encodings play a bigger role in algorithmic tasks. For example, in the *Add* and *AddNeg* tasks, models go from 0.005 and 0.042 accuracy to almost perfect accuracy (0.988 and 0.830 for the *rel2-e* model). Moreover tasks like SCAN or CFQ do not seem to be affected by position encodings, and using relative position encodings with only a bias term hurts in PCFG.

4.2 Decoder Type

Many tasks (such as the *duplication* or *PCFG* datasets used in our experiments) require models able to learn things like "output whatever is in position k of the input", rather than having to learn hard-coded rules for outputting the right token, depending on the input, a type of *symmetry* that can be captured with a *copy decoder*.

The copy decoder in our experiments is fairly simple, and works as follows (Figure 2, top-left). It assumes that the input and output vocabularies are the same (we use the union of input and output vocabularies in our experiments). For a given token x_i in the output (with final embedding y_i), in addition to the output probability distribution p_1 over the tokens in the vocabulary, the copy decoder produces a second distribution p_2, which is then mixed with p_1 via a weight w. p_2 is obtained by attending to the output of the last encoder layer (the attention *query* is calculated using a learnable weight matrix from y_i, the embeddings of the last encoder layer are used as the *keys*, and the *values* are a one-hot representation of the input tokens). The result is passed through a softmax layer, resulting in p_2.

Table 2 shows sequence-level classification accuracy for models with and without a copy decoder. As can be seen in the last column (*Avg.*), having a copy decoder consistently helps performance, with all models using a copy decoder (*abs-c*, *rel-eb-c* and *rel2-eb-c*) outperforming their counterparts without a copy decoder. Moreover, we see that the copy decoder helps the most in *PCFG* and *COGS*, while it does not seem to help in some other tasks.

Moreover, we would like to point out that there are other ways to set up copy decoders. For example Akyürek et al. (2021) propose defining a lexical

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg.
abs	0.005	0.042	0.000	0.000	0.000	0.500	0.000	0.003	0.174	0.434	0.177	0.304	0.137
rel-eb	0.003	0.011	0.486	0.444	0.000	0.500	0.089	**0.011**	0.257	0.452	0.249	0.290	0.233
rel2-eb	**0.978**	0.779	**0.737**	0.017	0.000	0.504	**0.091**	0.010	0.194	0.374	0.159	**0.311**	0.346
abs-c	0.006	0.021	0.000	0.000	0.000	0.501	0.000	0.003	0.230	0.390	**0.520**	0.301	0.164
rel-eb-c	0.004	0.007	0.271	0.460	0.000	0.413	0.026	0.009	**0.342**	0.541	0.474	0.311	0.238
rel2-eb-c	0.977	**0.791**	0.540	0.283	0.000	**0.528**	0.043	0.010	0.336	0.527	0.511	0.295	**0.403**

Table 2: Sequence-level accuracy with and without copy decoding (models with a copy decoder are marked with a "-c" suffix). Bolded numbers are the best results for each dataset in this table.

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg.
small-2	0.977	0.791	0.540	0.283	0.000	**0.528**	0.043	**0.010**	0.336	0.527	**0.511**	0.295	0.403
small-4	0.986	**0.835**	0.676	**0.572**	0.000	0.500	0.170	0.000	0.499	0.711	0.501	0.301	**0.479**
small-6	**0.992**	**0.835**	0.225	0.000	0.000	0.203	0.164	0.002	**0.548**	0.741	0.476	**0.312**	0.375
large-2	0.983	0.811	0.605	0.503	0.000	0.500	**0.184**	0.001	0.535	0.758	0.498	0.269	0.471
large-4	0.957	0.786	**0.684**	0.523	0.000	0.400	0.164	0.004	0.513	**0.770**	0.462	0.310	0.464
large-6	0.978	0.673	0.423	0.288	0.000	0.250	0.144	0.000	0.530	0.750	0.451	0.288	0.398

Table 3: Sequence-level accuracy for models of different sizes. All models are variations of the *rel2-eb-c* model in Table 2 (*small-2* is equivalent to *rel2-eb-c*). Bolded results represent the best results for each dataset in this table.

translation layer in the copy decoder, which allows models to translate tokens in the input to tokens in the output (which is useful in tasks such as SCAN, which have disjoint vocabularies). In their work, they propose to initialize this layer via a lexicon learning task.

4.3 Model Size

Next, we compare the effect of varying both the number of layers (l), as well as their size (d, f, h). Specifically, we tested models with number of layers l equal to 2, 4 and 6, and layers of two sizes: *small* ($d = 64$, $f = 256$, $h = 4$), and *large* ($d = 128$, $f = 512$, $h = 8$). We denote these models *small-2*, *small-4*, *small-6*, *large-2*, *large-4*, and *large-6*. All of the models in this section are variants of *rel2-eb-c*, our previous best (see Appendix C for parameter counts of our models).

Table 3 shows the sequence-level classification accuracy, showing a few interesting facts. First, in most algorithmic tasks, size does not help. Our hypothesis is that the logic required to learn these tasks does not require too many parameters, and large models probably overfit (e.g., like in *Duplication*)[3]. Some datasets, however, do benefit from size. For example, most *large* models outperform their respective *small* ones in both variants of PCFG. These results are not unexpected, as most compositional generalization datasets contain idealized examples, often generated via some form of grammar, and have very small vocabularies (see Table 7). Hence, models might not benefit from size as much as on complex natural language tasks.

4.4 Weight Sharing

In this section we evaluate the effect of sharing weights across transformer layers. When weight sharing is activated, all learnable weights from all layers in the *encoder* are shared across layers, and the same is true across the layers of the *decoder*.

Table 4 shows the resulting performance of the models (to be compared with Table 3). Surprisingly, weight sharing significantly boosts compositional generalization accuracy, and almost all models achieve a higher average accuracy across all datasets than their equivalent models in Table 3. In particular, datasets such as *AdditionNegatives* see a significant boost, with several models achieving higher than 0.9 accuracy (0.982 for *large-6s*). PCFG also significantly benefits from weight sharing, with the *large-6s* model achieving 0.634 and 0.828 in the productivity and systematicity versions, respectively. These are higher than previously reported results in the literature (using the original Transformer, which is a much larger model): 0.50 and 0.72 (Hupkes et al., 2020). Notice, moreover that achieving good results in PCFG (or SCAN) is easy with specialized models. The important achievement is doing so with general purpose models. Our hypothesis is that a model with shared weights across layers might have a more suited inductive bias to learn primitive operations that are applied repeatedly to the input of the transformer (copying, reversing, duplicating, etc.).

[3]Further investigation showed that lowering the learning rate improves performance in the larger models, preventing the phenomenon seen in the *Duplication* dataset. Systematically exploring this is left for future work.

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg.
small-2s	0.992	0.809	0.780	0.750	0.000	**0.699**	0.022	0.003	0.313	0.501	0.450	0.303	0.468
small-4s	0.991	0.955	0.708	0.580	0.000	0.500	0.172	**0.017**	0.534	0.723	0.445	0.292	0.493
small-6s	0.993	0.933	0.505	0.000	0.000	0.500	0.186	0.000	0.562	0.780	0.454	0.295	0.434
large-2s	**0.997**	0.894	**0.831**	0.848	0.000	0.584	0.033	0.002	0.511	0.638	0.465	0.292	0.508
large-4s	0.991	0.915	0.771	**0.882**	0.000	0.400	0.186	0.002	0.589	0.791	**0.475**	**0.327**	**0.527**
large-6s	0.985	**0.982**	0.241	0.000	0.000	0.500	**0.196**	0.000	**0.634**	**0.828**	0.454	0.303	0.427

Table 4: Sequence-level accuracy for all the models in Table 3, but sharing weights across layers.

4.5 Intermediate Representations

The key idea of an *intermediate representation* is to define a different representation of the target output that is easier to generate by the model, but that can be easily mapped to the desired output. Herzig et al. (2021) recently showed very promising results using this technique in several tasks. Defining useful intermediate representations is task-specific and not trivial. Thus we experimented with it in only two datasets: COGS and CFQ (Figure 3).

4.5.1 Intermediate Representation for COGS

Our intermediate representation for COGs turns the task from seq2seq into a *sequence tagging* task. We ask the model to produce 5 tags for each input token: a *parent*, the *role* of the relation between the token and its parent (if applicable), the *category*, the *noun determiner* (for nouns) and the *verb name* (for verbs). With these tags, the original output can be constructed deterministically. One of the main advantages of this is that the model is naturally pushed to produce outputs with the correct length even for longer inputs (improving *productivity*).

For the sequence tagging formulation, we used only the encoder part of the Transformer and added five prediction heads, to predict each tag. For *role*, *category*, *noun determiner* and *verb name*, we simply had a dense layer with a Sigmoid activation function. For the *parent* tag, we experimented with 3 different head types: *Absolute* used a dense layer with a Sigmoid activation to predict the absolute index of the parent in the input sequence (*-1* for no parent). *Relative* predicted the relative offset of the parent token with respect to the current token, or self for no parent. Finally, *Attention* used the attention weights from a new attention layer with 1 head to predict the parent.

Table 5 shows the experimental results comparing a few configurations of this new tagging approach to a few configurations of the seq2seq approach (see Appendix D for all other configurations). Examples in the structural generalization tasks are typically longer than in the training set and require *productivity*. All the models tested in the original COGS paper (Kim and Linzen, 2020) (and all of our seq2seq approaches above) achieved 0 accuracy in this category. The *small-6s* seq2seq model improves the overall performance from 0.278 to 0.475, but curiously has near 0 performance on *Verb Argument Structure Alternation* tasks, worse than the base *abs* model.

	seq2seq		tagging	
Model	abs	rel2-eb-c	abs	rel-eb
Size	small-2	small-6s	small-2	small-2s
Parent encoding			absolute	attention
Lexical Generalization: Primitives and Grammatical Roles				
Subject → Object (common noun)	0.309	0.899	0.911	0.969
Subject → Object (proper noun)	0.098	0.429	0.630	0.826
Object → Subject (common noun)	0.790	0.936	0.982	0.978
Object → Subject (proper noun)	0.207	0.951	0.993	0.995
Prim noun → Subject (common noun)	0.240	0.913	0.993	0.988
Prim noun → Subject (proper noun)	0.019	0.772	0.974	0.996
Prim noun → Object (common noun)	0.017	0.902	0.950	0.953
Prim noun → Object (proper noun)	0.000	0.513	0.651	0.700
Prim verb → Infinitival argument	0.000	0.766	0.000	0.001
Lexical Generalization: Verb Argument Structure Alternation				
Active → Passive	0.604	0.000	0.697	0.948
Passive → Active	0.196	0.001	0.535	0.897
Object-omitted transitive → Transitive	0.275	0.003	0.527	0.926
Unaccusative → Transitive	0.069	0.003	0.528	0.787
Double object dative → PP dative	0.819	0.000	0.590	0.958
PP dative → Double object dative	0.404	0.004	0.771	0.850
Lexical Generalization: Verb Class				
Agent NP → Unaccusative Subject	0.399	0.951	0.784	1.000
Theme NP → Obj-omitted trans Subj	0.688	0.965	0.791	0.701
Theme NP → Unergative subject	0.694	0.966	0.930	0.771
Structural Generalization: Phrases and Grammatical Roles				
Obj-mod PP → Subj-mod PP	0.000	0.000	0.000	0.299
Structural Generalization: Deeper Recursion				
Depth generalization: PP modifiers	0.003	0.000	0.138	0.681
Depth generalization: Sentential comp	0.000	0.000	0.000	0.233
Overall	0.278	0.475	0.637	0.784

Table 5: Sequence-level accuracy in different generalization subsets in COGS for both seq2seq and sequence tagging models. PP stands for prepositional phrase.

The intermediate representation based on tagging works much better. The base *abs* tagging model manages to get non-zero performance on one structural generalization task, which suggests that enforcing the right output length helps. Finally, when predicting the parent directly from attention weights, the structural generalization tasks score 0.2-0.7, compared to our previous near 0 scores (see Appendix D for common types of errors).

Overall, the sequence tagging intermediate representation achieves a much higher accuracy, with

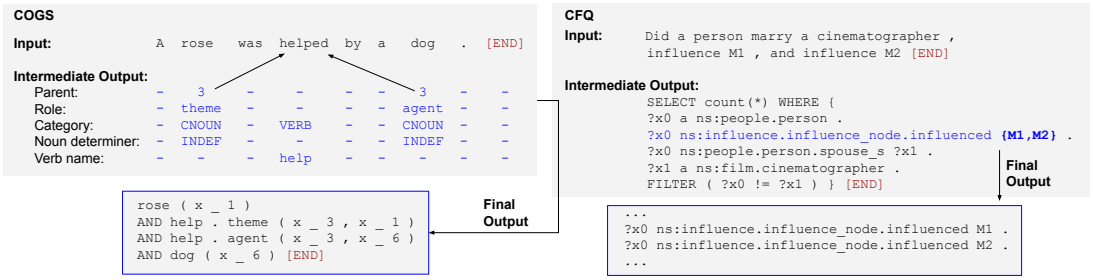

Figure 3: Examples from the intermediate representations for COGs and CFQ. For COGs, we framed the task as *sequence tagging* and made the model predict 5 tags for each token; for CFQ we compressed Cartesian products.

	CFQ	CFQ-im
abs	0.304	0.541
rel-eb	0.290	**0.555**
rel2-eb	0.311	0.541
rel-eb-c	0.311	0.541
rel2-eb-c	0.295	0.519
large-4	0.310	0.541
large-4s	**0.327**	0.474

Table 6: Sequence-level accuracy for different models for the original CFQ, and for CFQ with intermediate representations (*CFQ-im*). The top 5 models are small models with 2 layers, and the last four models are variants of *rel2-eb-c* (used in Tables 3 and 4).

one model reaching 0.784, higher than any previously reported performance in COGS in the literature, to the best of our knowledge. This suggests that the encoder has the power to parse the input correctly, but maybe the decoder is not capable of generating the correct output sequence from the encoder in the full transformer.

4.5.2 Intermediate Representation for CFQ

One of the difficulties in the CFQ dataset is that models need to learn to perform Cartesian products (e.g., for questions like "who directed and acted in M1 and M2?", the model needs to expand to "directed M1", "directed M2", "acted in M1" and "acted in M2"). However, as shown in our experiments above, this is a very hard task to learn. Hence, we followed the same idea as in Herzig et al. (2021), and defined an intermediate representation that removes the need to learn Cartesian products by allowing triples of the form *(entity list) - (relation list) - (entity list)*.

Table 6 shows the sequence-level classification accuracy for models on CFQ and on the version with intermediate representations (*CFQ-im*). While the different variations on Transformer models have little affect on the performance, the use of an intermediate representation significantly improves performance, going from around 0.3 accuracy for most Transformer models to over 0.5, and up to 0.555 for the *rel-eb* model. This is consistent with the results reported by Herzig et al. (2021).

5 Discussion

An overall trend is that algorithmic tasks seem to be greatly affected by the different architecture design decisions we explored. In all datasets, except for Cartesian product, there is at least one combination in our experiments that achieved high performance (close to 0.8 accuracy or higher). Cartesian products remain an open challenge for future work, where one of the big obstacles is learning to produce much longer outputs than seen during training (output is quadratic with respect to input size).

There are some datasets, such as *SCAN-aj*, where we did not see large improvements in performance. The main obstacle is learning to handle a symbol ("jump") having seen it very few times (or even just once) during training (this also happens in some types of generalization in COGS). None of the variations we experimented with were enough to handle this type of compositionality either.

In conclusion, we observed:

1. *relative position encodings* (when both embeddings and biases are used) seem to never be detrimental (they either provided gains, or did not affect). Results indicate this significantly helps in *productivity*. Moreover, for tasks where positional information is important (such as addition, or reversing), adding positional encodings to decoder2encoder attention provided significant benefits. Finally, as Table 1 shows, for relative position embeddings to be beneficial, using embeddings was necessary; only using relative position biases was not enough.

2. Adding a *copy decoder* was generally beneficial. We saw some occasional degradation in

some tasks (e.g., *Reverse*), but these are high variance tasks (see Table 10 in the Appendix), where results are more uncertain.

3. *Model size* in terms of embedding dimensions, helped generally. Going from 2 to 4 layers provided a slight benefit in general. Our experiments show going to 6 layers hurt performance, but as noted earlier, additional (unreported preliminary) experiments indicated larger models might need smaller learning rates, with which they also seem to improve performance (systematic exploration of this is future work).

4. *Weight sharing* seems to benefit in tasks where there are a clear set of primitives that have to be learned (PCFG in particular), or algorithmic tasks, but it seems to hurt in COGs. Hence, weight sharing does not provide general benefits as the previous modifications.

5. Intermediate representations, although dataset-specific, significantly help when they can be defined, as expected.

6 Conclusions

This paper presented an empirical study of the design space of Transformer models, evaluated in a collection of benchmarks for compositional generalization in language and algorithmic tasks. Our results show that, compared to a baseline Transformer, significant gains in compositional generalization can be achieved. Specifically, the baseline Transformer achieved an average sequence-level accuracy of 0.137, while we showed this can increase to up to 0.527 with some design changes. Accuracy levels of up to 0.493 can be achieved without increasing the parameter count of our baseline model (see Appendix C for parameter counts). Moreover, we achieved state-of-the-art results in COGS (at the time of submission), showing 0.784 accuracy on the generalization set, and two PCFG splits (0.634 and 0.828 respectively). This shows that a key factor in training models that generalize compositionally is to provide the right inductive biases.

As part of our future work, we want to explore more dimensions, such as pre-training and optimizer parameters, and study the implications of our results in compositional generalization in large models on real world tasks.

References

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284.

Ekin Akyürek and Jacob Andreas. 2021. Lexicon learning for few-shot neural sequence modeling. *arXiv preprint arXiv:2106.03993*.

Jacob Andreas. 2019. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150v1*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. *arXiv preprint arXiv:2108.12284*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *arXiv preprint arXiv:2104.07478*.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

Juyong Kim, Pradeep Ravikumar, Joshua Ainslie, and Santiago Ontañón. 2021. Improving compositional generalization in classification tasks via structure annotations. *arXiv preprint arXiv:2106.10434*.

Najoung Kim and Tal Linzen. 2020. Cogs: A compositional generalization challenge based on semantic interpretation. *arXiv preprint arXiv:2010.05465*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Adam Liška, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Sara Veldhoen, Dieuwke Hupkes, Willem H Zuidema, et al. 2016. Diagnostic classifiers revealing how neural networks process hierarchical structure. In *CoCo@ NIPS*, pages 69–77.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*.

Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. Hibert: Document level pre-training of hierarchical bidirectional transformers for document summarization. *arXiv preprint arXiv:1905.06566*.

A Implementation Details

We used a standard Transformer implementation[4], and added all the proposed variations on top of it. All experiments were run on machines with a single CPU and a single Tesla V100 GPU. All parameters were left to their default values from the original implementation, including the learning rate schedule (which could probably be further tweaked if state-of-the-art results are sought), as we were just aiming to compare inductive biases, rather than aim for SOTA results.

Additionally, we would like to highlight some implementation details, which surprisingly had large effects on our experimental results. *Layer normalization* operations in our Transformer implementation were done *after* each sublayer (attention and feed forward). Embedding layers were initialized with the Keras default "uniform" Keras initializer (uniform random distribution in the range $[-0.05, 0.05]$). Dense layers were initialized also with the Keras default Glorot initializer (uniform random distribution with mean 0 and standard deviation $\sqrt{2/(fan\_in + fan\_out)}$) (Glorot and Bengio, 2010). While these details might not seem that important, we were unable to reproduce some of the results reported above using a re-implementation of the Transformer model in Flax, which used different defaults (and layer normalization before each sublayer rather than after) unless we changed these implementation details to match those of the Keras implementation. This indicates that these low-level details also have an effect on the learning bias of the models, with an impact in compositional generalization, which we plan to study in the future.

B Detailed Results

Table 8 shows the average sequence-level accuracy for all the models evaluated in this paper, all in one table. We used the same names as used in the paper (as models *rel2-eb-c* and *small-2* both refer to the same model, we included the row twice, with both names, for clarity).

Table 9 shows the maximum accuracy each model achieved in each dataset out of the 3 to 10 repetitions we did for each dataset. Recall we used 3 repetitions for SCAN-l, SCAN-aj, PCFG-p, PCFG-s, COGS and CFQ, 5 repetitions for Add, AddNeg, Reverse, Dup and Cart, and 10 repetitions for Inters (as it was the dataset where we saw more extreme results). An interesting phenomenon observed in the *Inters* dataset is that models tend to achieve either random accuracy (around 0.5), or perfect accuracy (1.0). Very rarely models achieve intermediate values. This support the *needle-in-a-haystack* argument of Liška et al. (2018), who saw that while LSTMs have the capability of generalize compositionally, what happens in practice is that gradient descent has a very low probability of converging to weights that do so (finding the "compositional needle" in a haystack). We observed a similar thing in our experiments, but saw that some Transformer architectures resulted in an increased chance of finding this *needle*.

Table 10 shows the standard deviation in the sequence-level accuracy we observed in our experiments. As can be seen, the algorithmic tasks result in a much larger standard deviation. In some datasets (e.g., Add and Inters) it was common for morels to either achieve near 0% accuracy (50% in Inters) or near 100% accuracy, but few values in between.

C Parameter Counts

Table 11 shows the parameter count for all the models used in this paper, notice that exact parameter counts vary per dataset, as each dataset has a different token vocabulary, and hence both the token embedding and the output layers vary. One interesting result is that in our experiments, parameter count is not, by itself, sufficient to increase compositional generalization. Our best model overall (*large-4s*) only had about 0.5 million parameters, and outperformed significantly larger models. Another ex-

Dataset	$\lvert Train \rvert$	$\lvert Test \rvert$	$\lvert Vocab \rvert$	*Epochs*
Add	200000	1024	14	2
AddNeg	200000	1024	16	10
Reverse	200000	1024	14	2
Dup	200000	1024	14	4
Cart	200000	1024	24	4
Inters	200000	1024	106	8
SCAN-l	16989	3919	25	24
SCAN-aj	14669	7705	25	24
PCFG-p	81011	11331	537	20
PCFG-s	82167	10175	537	20
COGS	24155	21000	876	16
CFQ	95743	11968	184	16

Table 7: Size of the training/test sets, vocab and training epochs we used for the different datasets.

[4] https://www.tensorflow.org/tutorials/text/transformer

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg.
abs	0.005	0.042	0.000	0.000	0.000	0.500	0.000	0.003	0.174	0.434	0.177	0.304	0.137
rel-e	0.004	0.018	0.422	0.486	0.004	0.501	0.064	0.003	0.238	0.451	0.170	0.322	0.224
rel-b	0.002	0.005	0.277	0.362	**0.054**	0.501	0.049	0.007	0.042	0.102	0.126	0.276	0.150
rel-eb	0.003	0.011	0.486	0.444	0.000	0.500	0.089	0.011	0.257	0.452	0.249	0.290	0.233
rel2-e	0.988	0.830	0.787	0.010	0.000	0.501	0.032	0.007	0.159	0.353	0.259	0.322	0.354
rel2-b	0.140	0.708	0.056	0.253	0.000	0.504	0.080	0.002	0.041	0.117	0.138	0.319	0.197
rel2-eb	0.978	0.779	0.737	0.017	0.000	0.504	0.091	0.010	0.194	0.374	0.159	0.311	0.346
abs-c	0.006	0.021	0.000	0.000	0.000	0.501	0.000	0.003	0.230	0.390	0.520	0.301	0.164
rel-eb-c	0.004	0.007	0.271	0.460	0.000	0.413	0.026	0.009	0.342	0.541	0.474	0.311	0.238
rel2-eb-c	0.977	0.791	0.540	0.283	0.000	0.528	0.043	0.010	0.336	0.527	**0.511**	0.295	0.403
small-2	0.977	0.791	0.540	0.283	0.000	0.528	0.043	0.010	0.336	0.527	**0.511**	0.295	0.403
small-4	0.986	0.835	0.676	0.572	0.000	0.500	0.170	0.000	0.499	0.711	0.501	0.301	0.479
small-6	0.992	0.835	0.225	0.000	0.000	0.203	0.164	0.002	0.548	0.741	0.476	0.312	0.375
large-2	0.983	0.811	0.605	0.503	0.000	0.500	0.184	0.001	0.535	0.758	0.498	0.269	0.471
large-4	0.957	0.786	0.684	0.523	0.000	0.400	0.164	0.004	0.513	0.770	0.462	0.310	0.464
large-6	0.978	0.673	0.423	0.288	0.000	0.250	0.144	0.000	0.530	0.750	0.451	0.288	0.398
small-2s	0.992	0.809	0.780	0.750	0.000	**0.699**	0.022	0.003	0.313	0.501	0.450	0.303	0.468
small-4s	0.991	0.955	0.708	0.580	0.000	0.500	0.172	**0.017**	0.534	0.723	0.445	0.292	0.493
small-6s	0.993	0.933	0.505	0.000	0.000	0.500	0.186	0.000	0.562	0.780	0.454	0.295	0.434
large-2s	**0.997**	0.894	**0.831**	0.848	0.000	0.584	0.033	0.002	0.511	0.638	0.465	0.292	0.508
large-4s	0.991	0.915	0.771	**0.882**	0.000	0.400	0.186	0.002	0.589	0.791	0.475	**0.327**	**0.527**
large-6s	0.985	**0.982**	0.241	0.000	0.000	0.500	**0.196**	0.000	**0.634**	**0.828**	0.454	0.303	0.427

Table 8: Average sequence-level accuracy for all the models evaluated in this paper.

ample, of this is that the models with shared layer parameters outperform their counterparts without parameter sharing, although they naturally have less parameters.

D Detailed Results in COGS

Table 12 shows the results of some of the models we tested in the COGS dataset (including seq2seq and sequence tagging models), with the accuracy broken down by the type of example in the generalization set. The COGS dataset contains four splits: training, dev, test and generalization (generalization is the one used to measure compositional generalization, and the set reported in the main paper). All but one shown configuration achieve more than 95% sequence level accuracy on the test and development splits after training for 16 epochs over the training data. The generalization set is split into several generalization tasks as described above, to break down performance by type of generalization (overall performance in the generalization set is shown in the bottom row).

The best tagging model does much better than the base seq2seq model (0.784 vs. 0.278). Notably the tagging model does relatively well on the *Depth generalization: Prepositional phrase (PP) modifiers* task achieving accuracy 0.681. When the depth of the model is increased from 2 to 6, the score on this task increases from 0.681 to 0.819, i.e. the model with more layers can parse deeper recursion. However, increasing the encoder depth at the same time dramatically lowers the performance on *Verb Argument Structure Alternation* tasks.

Since many of the tasks are solved to near perfect accuracy, here we briefly discuss the types of the remaining errors. The one type of task where sequence tagging models did worse than seq2seq is *Prim verb* $\rightarrow$ *Infinitival argument*, which measures one shot generalization of an example with only a single verb to examples where the verb is used in sentences. The cause of this is that the tagging example with only a single verb doesn't actually encode the type of relations the verb allows, so the tagging model is actually not provided the full information in the only example for this one shot learning task. Nevertheless, this category was solved in our seq2seq models with a copy decoder.

Curiously, some errors, that the tagging model with *attention* in the parent prediction head makes, are quite quite reasonable. For example in the *Obj-mod PP* $\rightarrow$ *Subj-mod PP* task, the model often gets the complete parsing tree correctly, and the only error is the predicted relation of the subject to the predicate (instead of *agent* the model predicts *theme* as is present in all the training examples, where the prepositional phrase modifies the object).

Another task where even the best tagging model achieves a low score (0.233) is *Depth generalization: Sentential complements*. The examples in this task are long complex sentences chained together with the conjunction *that*. The most common error here is to predict that the main verb depends

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ
abs	0.008	0.131	0.002	0.000	0.000	0.500	0.000	0.008	0.191	0.462	0.211	0.326
rel-e	0.010	0.059	0.597	0.908	0.034	0.511	0.115	0.007	0.257	0.496	0.281	0.346
rel-b	0.004	0.016	0.331	0.417	**0.137**	0.510	0.072	0.013	0.047	0.112	0.170	0.305
rel-eb	0.006	0.018	0.658	0.795	0.001	0.502	0.129	0.023	0.268	0.528	0.306	0.333
rel2-e	**1.000**	0.943	0.917	0.038	0.000	0.512	0.058	0.018	0.182	0.457	0.332	0.357
rel2-b	0.256	0.910	0.132	0.339	0.002	0.529	0.116	0.004	0.049	0.137	0.187	0.342
rel2-eb	**1.000**	0.875	0.824	0.062	0.000	0.519	0.124	0.018	0.233	0.479	0.205	0.333
abs-c	0.021	0.037	0.000	0.000	0.000	0.506	0.000	0.005	0.250	0.420	0.550	0.312
rel-eb-c	0.006	0.027	0.504	0.721	0.000	**1.000**	0.031	0.021	0.361	0.562	**0.581**	0.351
rel2-eb-c	0.998	0.842	0.861	0.683	0.000	**1.000**	0.082	0.014	0.346	0.581	0.576	**0.369**
small-2	0.998	0.842	0.861	0.683	0.000	**1.000**	0.082	0.014	0.346	0.581	0.576	**0.369**
small-4	0.992	0.877	0.939	0.805	0.000	0.500	0.197	0.001	0.509	0.734	0.520	0.342
small-6	**1.000**	0.922	0.576	0.000	0.000	0.500	**0.199**	0.007	0.571	0.766	0.516	0.330
large-2	0.998	0.896	0.933	0.882	0.000	0.500	0.197	0.002	0.548	0.762	0.530	0.314
large-4	0.996	0.953	0.848	0.855	0.000	0.500	**0.199**	0.010	0.523	0.782	0.500	0.360
large-6	0.994	0.887	0.619	0.856	0.000	0.500	0.195	0.000	0.549	0.766	0.483	0.317
small-2s	0.998	0.871	**0.979**	**0.972**	0.000	**1.000**	0.044	0.006	0.328	0.519	0.487	0.348
small-4s	0.998	0.986	0.870	0.871	0.000	0.500	0.175	**0.039**	0.540	0.742	0.515	0.362
small-6s	**1.000**	0.984	0.821	0.000	0.000	0.500	**0.199**	0.000	0.569	0.788	0.486	0.344
large-2s	**1.000**	0.945	0.952	0.955	0.000	**1.000**	0.054	0.003	0.526	0.641	0.563	0.304
large-4s	**1.000**	0.959	0.923	0.959	0.000	0.500	0.195	0.004	0.604	0.810	0.481	0.362
large-6s	**1.000**	**0.998**	0.489	0.000	0.000	0.500	0.198	0.000	**0.642**	**0.832**	0.469	0.361

Table 9: Maximum sequence-level accuracy achieved in a given repetition for all the models evaluated in this paper.

on another verb far away in the sentence structure, instead of predicting that it has no parent. The distance to the incorrectly predicted parent is often more than 16, which was the limit on our relative attention offsets. The attention mechanism seems to get confused by seeing many more tokens in this test split than during training.

E Dataset Details

This appendix presents more details on the datasets used in this paper, as well as on the type of compositionality involved in each of them.

- **Addition** (*Add*): This is a synthetic addition task, where the input contains the digits of two integers, and the output should be the digits of their sum. The training set contains numbers with up to 8 digits, and the test set contains numbers with 9 or 10 digits. Numbers are padded to reach a length of 12 so that it's easy to align the digits that need to be added. We found that without padding, the task became much harder. **Types of compositionality:** models need to learn that there is a primitive operation "adding two digits (with carry)" that is repeatedly applied at each position. Models that learn position-specific shortcuts will not generalize to longer input lengths (as they would have learned no rules to produce the most significant digits, which would have never been seen during training). This mostly corresponds to *productivity* type of compositional generalization.

- **AdditionNegatives** (*AddNeg*): The same as the previous one, but 25% of the numbers are negative (preceded with the "−" token). **Types of compositionality:** the type of compositionality requires by this task is similar to that of the previous task, except that the general rules that need to be learned (independent of position) are more complex due to negative numbers. So, the model needs to learn three basic primitive operations that are the same irrespective of the position of the digits: "add two digits with carry", "subtract first from second with carry", and "subtract second from first with carry", and learn when to apply each. This also mostly corresponds to *productivity* type of compositional generalization.

- **Reversing** (*Reverse*): Where the output is expected to be the input sequence in reverse order. Training contains sequences of up to 16 digits, and the test set contains lengths between 17 to 24. **Types of compositionality:** the difficult part of this task is to learn to reverse position embeddings in a way that generalizes to longer inputs than seen during training, in order to attend and produce the right

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ
abs	0.003	0.047	0.001	0.000	0.000	0.000	0.000	0.004	0.014	0.039	0.067	0.022
rel-e	0.003	0.017	0.169	0.271	0.012	0.003	0.045	0.004	0.023	0.078	0.103	0.027
rel-b	0.002	0.006	0.078	0.046	0.073	0.003	0.038	0.006	0.005	0.014	0.040	0.025
rel-eb	0.002	0.007	0.211	0.287	0.000	0.001	0.038	0.011	0.013	0.066	0.050	0.047
rel2-e	0.009	0.074	0.167	0.014	0.000	0.004	0.023	0.009	0.016	0.111	0.104	0.035
rel2-b	0.122	0.202	0.051	0.055	0.001	0.009	0.039	0.002	0.011	0.018	0.045	0.016
rel2-eb	0.029	0.067	0.057	0.024	0.000	0.007	0.029	0.008	0.047	0.101	0.043	0.020
abs-c	0.009	0.010	0.000	0.000	0.000	0.003	0.000	0.002	0.024	0.027	0.038	0.013
rel-eb-c	0.003	0.011	0.135	0.157	0.000	0.322	0.005	0.011	0.017	0.036	0.093	0.025
rel2-eb-c	0.035	0.053	0.208	0.289	0.000	0.239	0.033	0.005	0.009	0.048	0.056	0.063
small-2	0.035	0.053	0.208	0.289	0.000	0.239	0.033	0.005	0.009	0.048	0.056	0.063
small-4	0.004	0.054	0.213	0.184	0.000	0.000	0.046	0.000	0.010	0.019	0.028	0.049
small-6	0.007	0.120	0.233	0.000	0.000	0.256	0.056	0.004	0.024	0.026	0.047	0.022
large-2	0.016	0.074	0.240	0.289	0.000	0.000	0.022	0.001	0.012	0.004	0.042	0.033
large-4	0.075	0.106	0.178	0.190	0.000	0.211	0.049	0.006	0.009	0.010	0.033	0.047
large-6	0.023	0.377	0.119	0.356	0.000	0.264	0.045	0.000	0.018	0.014	0.029	0.022
small-2s	0.007	0.038	0.255	0.254	0.000	0.346	0.021	0.003	0.014	0.019	0.054	0.039
small-4s	0.009	0.055	0.118	0.261	0.000	0.000	0.005	0.020	0.008	0.023	0.068	0.054
small-6s	0.012	0.047	0.208	0.000	0.000	0.001	0.017	0.000	0.006	0.007	0.030	0.041
large-2s	0.004	0.031	0.131	0.167	0.000	0.156	0.027	0.001	0.018	0.004	0.102	0.011
large-4s	0.007	0.039	0.127	0.066	0.000	0.211	0.016	0.002	0.015	0.017	0.009	0.043
large-6s	0.020	0.015	0.159	0.000	0.000	0.000	0.002	0.000	0.008	0.007	0.013	0.037

Table 10: Standard deviation of the sequence level accuracy results.

output sequences. This mostly corresponds to *productivity* type of compositional generalization, as the model needs to learn to reverse position embeddings for longer sequences than seen during training.

- **Duplication** (*Dup*): The input is a sequence of digits and the output should be the same sequence, repeated twice. Training contains sequences up to 16 digits, and test from 17 to 24. **Types of compositionality:** Learning to repeat the input several times is not a particularly hard task for a Transformer, but we noticed that the difficult part was learning when to stop producing output (exactly after repeating the input twice in this case). This problem was also noted in the work of (Csordás et al., 2021), and mostly corresponds to *productivity* type of compositional generalization.

- **Cartesian** (*Cart*): The input contains two sequences of symbols, and the output should be their Cartesian product. Training contains sequences of up to 6 symbols (7 or 8 for testing). **Types of compositionality:** this is a very challenging task that requires very demanding *productivity*, as the model needs to learn to learn to compose the basic operation of pairing elements from both sets via two nested loops: iterating over each of the two input sets.

- **Intersection** (*Inters*): Given two sequences of symbols, the output should be whether they have a non-empty intersection. Training contains sets with size 1 to 16, and testing 17 to 24. **Types of compositionality:** the main challenge in this dataset is to learn short-cut rules such as "if the first set contains `a4` and the second set also contains `a4` then the output should be `true`". However, the model needs to learn to ignore these token specific rules, and learn the general rule of finding two identical tokens regardless of which specific token they are, which could be seen as a form of *systematicity*. Moreover, this needs to be learned in a way that generalizes to longer inputs (*productivity*).

- **SCAN-length** (*SCAN-l*): The *length split* of the SCAN dataset (Lake and Baroni, 2018). The SCAN dataset asks the model to learn to interpret and execute natural language instructions with a limited vocabulary. For example, if the input is "walk twice", the output should be "I_WALK I_WALK". There are a set of primitive actions (walk, jump, etc.), and a set of modifiers (twice, thrice, left, etc.) and composition operators (e.g., and), and the model needs to learn how to compose and execute all of those instructions to generate the output sequence. In this specific *length split*, the training and test sets are split by length (the

	Add	AddNeg	Reverse	Dup	Cart	Inters	SCAN-l	SCAN-aj	PCFG-p	PCFG-s	COGS	CFQ
abs	236k	236k	236k	236k	238k	253k	238k	238k	337k	337k	402k	268k
rel-e	239k	239k	239k	239k	241k	257k	241k	241k	340k	340k	405k	272k
rel-b	236k	236k	236k	236k	238k	254k	238k	238k	337k	337k	402k	269k
rel-eb	239k	239k	239k	239k	241k	257k	241k	241k	340k	340k	405k	272k
rel2-e	239k	239k	239k	239k	241k	257k	241k	241k	340k	340k	405k	272k
rel2-b	236k	236k	236k	236k	238k	254k	238k	238k	337k	337k	402k	269k
rel2-eb	239k	239k	239k	239k	241k	257k	241k	241k	340k	340k	405k	272k
abs-c	241k	241k	241k	241k	242k	258k	243k	243k	341k	341k	407k	273k
rel-eb-c	243k	244k	243k	243k	245k	261k	245k	245k	344k	344k	410k	276k
rel2-eb-c	243k	244k	243k	243k	245k	261k	245k	245k	344k	344k	410k	276k
small-2	243k	244k	243k	243k	245k	261k	245k	245k	344k	344k	410k	276k
small-4	480k	480k	480k	480k	482k	498k	482k	482k	581k	581k	646k	513k
small-6	717k	717k	717k	717k	719k	735k	719k	719k	818k	818k	883k	750k
large-2	1.88m	1.88m	1.88m	1.88m	1.88m	1.92m	1.88m	1.88m	2.08m	2.08m	2.21m	1.95m
large-4	1.88m	1.88m	1.88m	1.88m	1.88m	1.92m	1.88m	1.88m	2.08m	2.08m	2.21m	1.95m
large-6	2.81m	2.81m	2.81m	2.81m	2.81m	2.84m	2.81m	2.81m	3.01m	3.01m	3.14m	2.87m
small-2s	125k	125k	125k	125k	127k	143k	127k	127k	226k	226k	291k	158k
small-4s	125k	125k	125k	125k	127k	143k	127k	127k	226k	226k	291k	158k
small-6s	125k	125k	125k	125k	127k	143k	127k	127k	226k	226k	291k	158k
large-2s	486k	487k	486k	486k	490k	521k	490k	490k	687k	687k	818k	552k
large-4s	486k	487k	486k	486k	490k	521k	490k	490k	687k	687k	818k	552k
large-6s	486k	487k	486k	486k	490k	521k	490k	490k	687k	687k	818k	552k

Table 11: Parameter counts for the models used in this paper.

test set contains the longest sequences and the training set the shortest ones). **Types of compositionality:** Overall, SCAN requires significant *systematicity* to be solved, and this split in particular focuses on *productivity*.

- **SCAN-add-jump** (*SCAN-aj*): The *add primitive jump split* of the SCAN dataset (Lake and Baroni, 2018). In this split, the "jump" instruction is only seen during training in isolation (i.e., there is a training example "jump" $\rightarrow$ "I_JUMP"), but the test set contains this instruction heavily, and in combination with other constructs. **Types of compositionality:** this split in particular focuses more on *systematicity*.

- **PCFG-productivity** (*PCFG-p*): The productivity split of the PCFG dataset (Hupkes et al., 2020). The PCFG dataset is a synthetic dataset where each example contains a set of operations that need to be done to one or more input strings, and the model needs to learn to apply these operations and produce the final output. Operations include reversing, duplicating, getting the first element, etc. **Types of compositionality:** this split in particular focuses on *productivity*, as test examples contain longer sequences of instructions than those seen during training.

- **PCFG-sytematicity** (*PCFG-s*: The systematicity split of the PCFG dataset (Hupkes et al., 2020). **Types of compositionality:** this split focuses on *systematicity*, by testing the model recombining operations in ways never seen during training.

- **COGS**: The generalization split of the COGS semantic parsing dataset (Kim and Linzen, 2020). This is a semantic parsing dataset, where the input is a sentence in natural language, and the output should be a logical representation of the sentence. **Types of compositionality:** The generalization split contains combinations not seen during training, while most of these focus on *systematicity* (e.g., constructions that had only been seen as subjects, now they are seen as objects), some part of the test set focuses on *productivity* (having deeper nesting of propositional phrases, for example). This, productivity type of generalization, is where our sequence tagging approach significantly outperforms previous approaches.

- **CFQ-mcd1** (*CFQ*): The MCD1 split of the CFQ dataset (Keysers et al., 2019). This dataset asks a model to learn how to translate delexicalized natural language queries into SPARQL. **Types of compositionality:** the MCD1 split of this dataset focuses specifically

on *systematicity*, but more concretely, there are two additional ways in which this dataset is hard compositionally. First, solving this dataset requires solving Cartesian products (which is the reason for which we added the separate Cartesian product task), since some question contains constructions like: "Who directed, played and produced movies M1, M2 and M3", which get translated into 9 SPARQL clauses (the Cartesian product). Second, SPARQL clauses are supposed to be produced in alphabetical order, hence the model needs to learn how to sort.

Finally, table 7 shows the size of the training and test sets for each dataset, as well as the size of their vocabularies. For the vocabulary, we used the union of the input and output vocabularies as a unified vocabulary. We also show the number of training epochs we performed in each dataset (this was chosen as the number after which performance stabilized with some initial models; it was not tuned afterwards during the systematic evaluation presented below).

	seq2seq						tagging											
Model	abs	abs	rel2-eb	rel2-eb	rel2-eb-c	rel2-eb-c	abs	abs	abs	abs	abs	abs	rel-eb	rel-eb	rel-eb	rel-eb	rel-eb	rel-eb
Size	small-2	small-6	small-2s	small-6s	small-2s	small-6s	small-2	small-6	small-2	small-6	small-2	small-6	small-2s	small-6s	small-2s	small-6s	small-2s	small-6s
Parent encoding							absolute	absolute	relative	relative	attention	attention	absolute	absolute	relative	relative	attention	attention
Test split	0.981	0.646	0.978	0.961	0.983	0.974	0.997	0.994	0.997	0.997	0.997	0.997	0.995	0.997	1.000	1.000	1.000	1.000
Dev split	0.976	0.625	0.968	0.950	0.976	0.974	0.996	0.994	0.997	0.996	0.997	0.996	0.995	0.996	1.000	1.000	1.000	0.999
Lexical Generalization: Novel Combination of Familiar Primitives and Grammatical Roles																		
Subject → Object (common noun)	0.309	0.008	0.030	0.011	0.900	0.899	0.911	0.938	0.914	0.916	0.893	0.918	0.899	0.972	0.978	0.996	0.969	0.956
Subject → Object (proper noun)	0.098	0.000	0.000	0.000	0.581	0.429	0.630	0.590	0.563	0.494	0.731	0.610	0.690	0.671	0.567	0.580	0.826	0.672
Object → Subject (common noun)	0.790	0.091	0.304	0.175	0.959	0.936	0.982	0.973	0.974	0.994	0.965	0.935	0.945	0.988	0.992	0.999	0.978	0.769
Object → Subject (proper noun)	0.207	0.007	0.023	0.019	0.970	0.951	0.993	0.986	0.995	0.991	0.993	0.990	0.985	0.847	0.998	0.999	0.995	0.984
Primitive noun → Subject (common noun)	0.240	0.098	0.242	0.216	0.956	0.913	0.993	0.983	0.995	0.972	0.990	0.978	0.927	0.976	1.000	1.000	0.988	0.927
Primitive noun → Subject (proper noun)	0.019	0.004	0.016	0.017	0.422	0.772	0.974	0.983	0.979	0.985	0.993	0.992	0.796	0.990	0.959	0.803	0.996	0.999
Primitive noun → Object (common noun)	0.017	0.007	0.014	0.012	0.929	0.902	0.950	0.968	0.939	0.945	0.949	0.972	0.904	0.986	0.996	1.000	0.953	0.966
Primitive noun → Object (proper noun)	0.000	0.000	0.000	0.000	0.200	0.513	0.651	0.687	0.557	0.569	0.722	0.649	0.624	0.676	0.545	0.467	0.700	0.673
Primitive verb → Infinitival argument	0.000	0.000	0.000	0.000	0.476	0.766	0.000	0.000	0.000	0.000	0.011	0.000	0.017	0.000	0.000	0.000	0.001	0.000
Lexical Generalization: Verb Argument Structure Alternation																		
Active → Passive	0.604	0.107	0.147	0.000	0.000	0.000	0.697	0.122	0.741	0.160	0.736	0.004	0.210	0.013	0.612	0.001	0.948	0.000
Passive → Active	0.196	0.067	0.002	0.001	0.001	0.001	0.535	0.115	0.617	0.014	0.625	0.163	0.539	0.000	0.586	0.073	0.897	0.000
Object-omitted transitive → Transitive	0.275	0.002	0.002	0.001	0.001	0.003	0.527	0.100	0.620	0.031	0.413	0.092	0.356	0.000	0.447	0.027	0.926	0.000
Unaccusative → Transitive	0.069	0.000	0.002	0.001	0.002	0.003	0.528	0.144	0.295	0.000	0.457	0.000	0.301	0.000	0.326	0.005	0.787	0.005
Double object dative → PP dative	0.819	0.005	0.095	0.001	0.002	0.000	0.590	0.279	0.778	0.345	0.732	0.037	0.424	0.000	0.853	0.018	0.958	0.000
PP dative → Double object dative	0.404	0.002	0.053	0.003	0.006	0.004	0.771	0.542	0.617	0.169	0.895	0.042	0.616	0.008	0.717	0.069	0.850	0.139
Lexical Generalization: Verb Class																		
Agent NP → Unaccusative Subject	0.399	0.000	0.003	0.002	0.958	0.951	0.784	0.661	0.951	0.911	0.952	0.949	0.982	0.980	0.995	0.999	1.000	0.999
Theme NP → Object-omitted transitive Subject	0.688	0.000	0.023	0.000	0.818	0.965	0.791	0.644	0.642	0.861	0.473	0.420	0.537	0.831	0.984	0.903	0.701	0.485
Theme NP → Unergative subject	0.694	0.000	0.023	0.000	0.843	0.966	0.930	0.715	0.643	0.858	0.544	0.570	0.583	0.895	0.960	0.919	0.771	0.530
Structural Generalization: Novel Combination Modified Phrases and Grammatical Roles																		
Object-modifying PP → Subject-modifying PP	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.029	0.000	0.000	0.000	0.000	0.299	0.371
Structural Generalization: Deeper Recursion																		
Depth generalization: PP modifiers	0.003	0.000	0.000	0.000	0.000	0.000	0.138	0.074	0.231	0.191	0.133	0.000	0.000	0.010	0.669	0.775	0.681	0.819
Depth generalization: Sentential complements	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.017	0.005	0.000	0.000	0.000	0.000	0.282	0.164	0.233	0.133
Overall	0.278	0.019	0.047	0.022	0.430	0.475	0.637	0.500	0.622	0.496	0.629	0.445	0.540	0.469	0.689	0.514	0.784	0.497

Table 12: Sequence-level accuracy in different subsets of the generalization set in COGS for both seq2seq and sequence tagging models (averaged over 5 runs). PP stands for prepositional phrase.