Generalization without systematicity: A reproduction study and extension into in-context learning

Axel Højmark
axho@di.ku.dk

Abstract

In this paper we aim to reproduce the findings from Lake and Baroni (2017), and go beyond the realm of RNN's with in-context-learning. We are able to reproduce their findings for experiment 1, 2 and 3 on all accounts, except for their models using attention. We make several test to verify that our implementation is correct, and attribute the difference in performance to their model selection approach. Using smaller OPT language models we outperform the RNNs by a large margin on experiment 3. However, when examining the OPTs further, we do not find evidence of true systematic compositionality on the SCAN task.

1 Introduction

The main goal of the Lake and Baroni (2017) paper was to better understand to what degree neural networks are capable of *systematic compositionality*. Systematic compositionality is defined as the algebraic capacity to understand and produce a potentially infinite number of novel combinations from known components (Chomsky, 1957).

To probe this, they introduced the SCAN dataset. The dataset consists of pairs of natural language and corresponding sequences of actions. The natural language commands are defined in a non-recursive grammar, which produces a total of 20.910 unambigous commands[1]. They then propose 3 experiments that test systematic compositionality utilizing the SCAN dataset. On these experiments they applied a range of sequence to sequence recurrent neural networks: SRNs (Elman, 1990), LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Chung et al., 2014). This also included recurrent networks augmented with an attentional mechanism, as described in Bahdanau et al. (2014).

[1]We refer to appendix A for examples from the SCAN dataset

In this paper we aim to reproduce the results for the best-overall architecture, and the best models from Experiments 1, 2 and 3. To be confident in our results, we also recreate the associated error analysis.
Furthermore, we wish to extend the SCAN generalization measures to modern language models, and examine the OPT model suite. The **O**pen **P**re-trained **T**ransformers are a collection of decoder-only pretrained transformers ranging from 125M to 175B parameters (Zhang et al., 2022). They were trained to roughly match the performance and sizes of the GPT-3 class of models (Brown et al., 2020), while also applying the latest best practices in data collection and efficient training. We found this set of models fascinating to study, after the recent rise of ChatGPT - a chatbot built upon GPT-3. Since GPT-3 and its relatives are not publicly available, we study OPT as a proxy.

2 Reproduction

Whenever specifics of the Lake and Baroni (2017) paper where vague, we turned to the PyTorch Seq2seq tutorial referenced by the authors[2]. All the results and graphs presented below will be averages over 5 independent runs. The tables will report the SD across the runs, and the charts will show ± 1 SEM as black bars. All charts are available in an upscaled format in appendix B. The model architectures vary across the experiments. These can be found in appendix C together with the training specifics in appendix D.

2.1 Experiment 1

In Experiment 1 the SCAN tasks are randomly split into a training set (80%) and a test set (20%). This tests the models abilities to decompose and

[2]The tutorial can be found here.

recombine commands seen during training in order to interpret novel ones.

	Reimplementation	Reported
Overall	98.7 ± 1.1	99.8
Best	99.8 ± 0.18	99.8

Table 1: Comparison of reported accuracies on the test set and our findings for experiment 1.

Table 7 shows a summary of our results. The accuracies are almost identical, and in both cases the reported accuracy lie within one standard deviation.
We proceed to test the overall-best network whilst varying the number of distinct training examples.

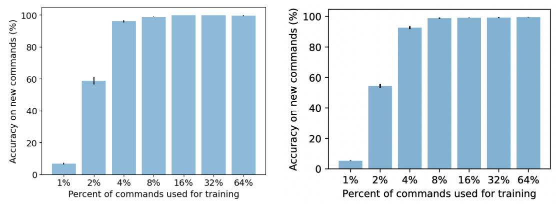


Figure 1: Comparison of reproduction (left) and reported (right) zero-shot generalization after training on a random subset of the SCAN tasks.

The two graphs are practically identical, showing the same pattern in accuracy and SEM across splits.

2.2 Experiment 2

In experiment 2 the goal is to test generalization across sequence lengths. The training set now contains all commands with 22 or fewer output actions. The test set includes all remaining commands with action sequences of lengths 24 - 48. In order for the model to succeed it must master verbs, modifiers and conjunctions to generate longer action sequences.

	Reimplementation	Reported
Overall	14.6 ± 1.8	13.8
Best	13.2 ± 4.6	20.8

Table 2: Overall findings for experiment 2

We outperform the reported overall-best accuracy by a small margin, but not by more than a standard deviation. However the best model for experiment 2 severely under performs the reported model by as much as 7.6 percentage points. We mainly attribute this to the authors model selection approach. This is elaborated in section 3.

We further examine accuracy across command and action sequence lengths for the overall-best model. In figure 2 one witnesses the same overall patterns in accuracy, showing that the partial success is almost entirely explained by generalization to the shortest action sequence lengths in the test set.

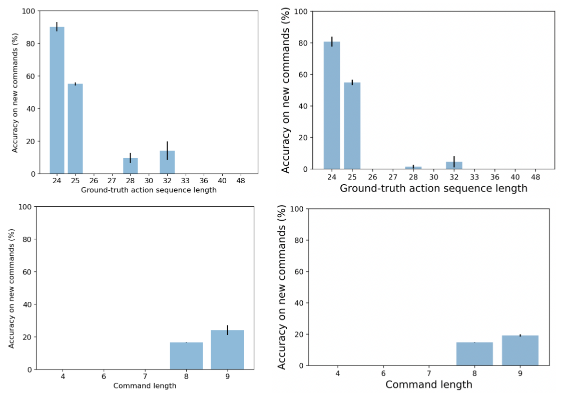


Figure 2: Top: accuracy distribution by action sequence length. Bottom: accuracy distribution by command length. Left: reproduction. Right: reported.

Additionally we studied whether the difficulty with long output sequences can be mitigated if the true length was provided by an oracle at evaluation. This can be found in appendix E. The same pattern persists: The reproduced overall-best is slightly more accurate, whilst the attention-based best model lags behind.

We also analysed the models for search related errors. As reported in the paper, we found no such examples. In appendix F we have provided a table showing the mean log probabilities of the golden sequence (target output) and the predicted ones.

2.3 Experiment 3

For the third experiment a single primitive command is selected. One then removes all composite commands containing the primitive from the training set. At test time the model is evaluated on the excluded composite commands. In table 3 you will find our results for the primitives `JUMP` and `LTURN`.
Once more, the overall-best models achieve similar performance and the best models (which utilize attention) are not scoring comparably.
We also test zero-shot generalization after adding varying amounts of compositional "jump" com-

	Reproduction	Reported
Overall `LTURN`	$87.3\% \pm 8.4$	90%
Best `LTURN`	$51.2\% \pm 9.4$	90.3%
Overall `JUMP`	$0.07\% \pm 0.05$	0.08%
Best `JUMP`	$0.01\% \pm 0.01$	1.2%

Table 3: Overall findings for experiment 3

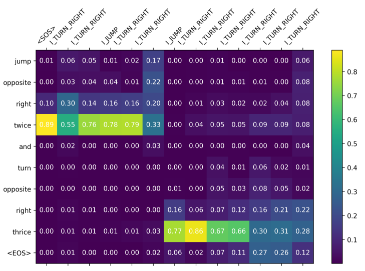


Figure 4: The attention map was produced using the best model architecture with the highest accuracy from experiment 2.

mands to the training set (see figure 3). We see, as in the paper, that performance rapidly grows when exposure to composed jump commands increases.

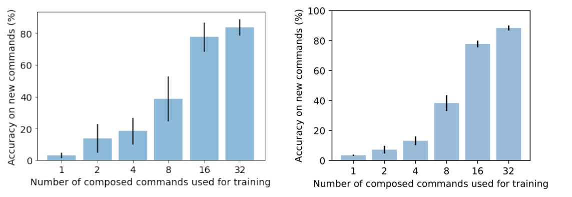


Figure 3: Comparison of reproduction (left) and reported (right) zero-shot generalization when presenting compositional "jump" commands during training. Accuracies are obtained from retrained versions of the `JUMP` best

Lastly we inspect the learned representations of the median overall-best model[3]. These can be found in appendix G. We observe similar closest primitives and composites in representational space, altough in a slightly different orderings. That is to be expected as most of the primitives, like `WALK` and `LOOK`, are distributionally identical.

3 Discussion of Reproduction Results

A clear pattern emeges from the reimplementation of experiment 1, 2 and 3. The models relying on the attention mechanism are systematically less accurate than reported. The question then becomes: Is the attention mechanism implemented correctly?[4]

To answer this, we first inspect the attention maps. It could indicate an inaccuracy in our implementation if the attention is spread uniformly, or is randomly scattered across the input sequence.

Looking at figure 4 we see a very different picture. In this example, the model clearly attends to the modifier "twice", for the entire duration it is needed for command generation. The moment that the first segment of the phrase is completed, it shifts to the "thrice" modifier, and largely ignores the now irrelevant initial part of the instruction. We inspected several attention maps, and none of them appeared nonsensical.

Nontheless the performance gap is still present. This brings us to the model selection approach used by Lake and Baroni (2017). They performed a grid search over 180 different architectures, each trained and tested on every experiment, choosing the models with the highest test accuracy[5]. Note that this introduces a bias in the test scores, as you are essentially overfitting on the test data across models. The way you normally mitigate this, is to reserve a third dataset, which is only used *once* after model selection. This will yield an unbiased estimate of the true loss.

In order to exemplify this experimentally we ran the best model from experiment 2, 20 more times, totaling 25 runs[6]. This new collection of runs ranged in accuracy all the way from 6.6% to 46.7%! If we now take the average of the 5 best runs, we get an accuracy of 29.2% beating the paper by 8.4 percentage points. This shows two things: 1. Our attention-based networks are more than capable of successfully modelling the task. That would likely not be the case for an erroneous implementation. 2. It illustrates the bias introduced when performing model selection over test data.
The points made above govern all experiments, and are not local to experiment 2. But this still leaves the question as to why this phenomena seemingly only affects the attention models. We hypothesize that the models relying on attention are somehow

[3]The paper was not particularly clear on which model they utilized for their encodings.

[4]We refer to appendix H for a complete overview over how the attention was integrated with the recurrent networks.

[5]See appendix I for the explored hyperparameters.

[6]The list of the 25 scores can be found in appendix J.

intrinsically more sensitive to initial parameters or the data sampled during training. This could be a result of the higher parameter count, although a deeper investigation would be necessary for a conclusive answer.

4 Beyond RNNs: In-context Learning

In the monumental paper "Language Models are Few-Shot Learners" the authors show that as large language models (LLM) grow in size, they can achieve competitive results compared to fully-supervised, fine-tuned language models when primed with only a handful of training samples (Brown et al., 2020). These results, and many other like them, indicate clear signs of systematic compositionality. In experiment 3, we witnessed how the recurrent models were practically incapable of generalizing to composed commands for the primitive "jump". As this was by far the hardest SCAN task proposed, we wanted to investigate how LLMs would perform.

As mentioned we will be studying the OPT suite of models. Due to hardware constraints, we limit ourselves to the 6.7B parameter model and its smaller relatives.

4.1 Prompt Design

For the first experiment we aimed to provide the model with the best possible context for solving the composition. When generating a prompt with K examples, we would extract the $K-1$ most similar training commands in relation to the evaluation command. The K'th example would be the isolated `JUMP` primitive. An example of such a prompt can be seen in figure 5.

	run and run = `RUN RUN`
	run and look = `RUN LOOK`
Context $\Rightarrow$	run and walk = `RUN WALK`
	jump = `JUMP`
	run and jump =
Target $\Rightarrow$	`RUN JUMP`

Figure 5: Illustrative *BLEU* prompt with $K = 4$. The first three examples have the highest BLEU similarity to the evaluation command. The last example is the jump primitive.

We used BLEU score a, method traditionally used for evaluating the performance of text generation models (Papineni et al., 2002), as a similarity measure between the evaluation command and the commands in the training set. BLEU works by comparing the n-grams of the generated text (training command) to the n-grams in the reference translations (evaluation command). This works surprising well in the SCAN domain as the n-grams effectively capture the structure of the underlying phrase-grammar[7].

We experimented with adding natural langauge instructions to the prompts such as "Given these examples: [examples] Translate the sentence: [command]" and several other variations. LLMs have been shown to be very sensitive to minor changes in prompts such as the example ordering and word choice (Lu et al., 2021). Preliminary tests showed that adding instructions in natural language had little impact on accuracy. They were therefore left out in favor of simplicity.

4.2 Results

For the following tasks we tested on a subset of 1000 out of the 7706 test samples available for the `JUMP` primitive task. This reduction was done in interest of compute time. The 1000 samples extracted, where of the shortest commands lengths. As we are limited by OPT's windows size of 2048 tokens, these samples accommodate longer prompts with more examples.

K / Model	350M	1.3B	2.7B	6.7B
2	0%	2.9%	**9.8%**	9.6%
4	6.8%	20.3%	**40.3%**	31.2%
8	0.8%	5.5%	**28.1%**	27.6%
16	0.2%	0.6%	7.7%	**8.6%**

Table 4: Accuracy for few-shot configurations of the *BLEU* prompts. The bold values are the highest accuracy for the row (K fixed).

In table 4 you can see the accuracy of different model sizes evaluated on various values of K. There are two immediate observations to be made:
1. Accuracy seems to maximized for $K = 4$. Any more examples rapidly decreases performance.
2. Model size has a large influence on performance. However, the largest model does not achieve the highest accuracy.

[7]One could alternatively have used the hidden representations from one of the encoder RNNs in conjunction with cosine similarity to find closely related samples. We favoured BLEU for its speed and explainability.

These two observations are particularly interesting as they go against all the major results in the literature. We hypothesize that this happens because the models are not utilizing a deep understanding of the SCAN domain to solve the task. If this were the case, an increase of examples should strengthen its model of the underlying grammar, leading to higher accuracy. Instead, what the model is likely doing, is simply inserting the `JUMP` primitive in place of some other primitive. In this case, having access to more examples would not be beneficial, as it would dilute the pool of correct templates.
This also helps explain why the largest model is not the most performant. For the simple insertion intrepretation, it is likely not necessary to have more than 2.7B parameters, leaving the ordering of the two mostly to chance. That being said, i would assume, as shown in the original OPT paper, that across multiple SCAN tasks the 6.7B model would be dominant (Zhang et al., 2022).

In order to test the hypothesis that the model relies on the much simpler task of inserting `JUMP`, rather than learning the underlying grammar, we next test the models on random samples from the training data. The first $K - 1$ examples are now randomly drawn from the training set, whilst the K'th example is still the isolated primitive[8]. We also extended K up to 32, in order to give a better chance at grasping the verbs, modifiers and conjunctions. For the model to perform well on this task, it must infer the phrase-structure grammar from the examples, rather than copying and editing an already present sequence.

K / Model	350M	1.3B	2.7B	6.7B
2	0%	0%	0.2%	**0.4%**
4	0%	0%	0%	**0.3%**
8	0%	0%	0%	**0.3%**
16	0%	0%	0%	**0.1%**
32	0%	0%	0%	**0.2%**

Table 5: Accuracy for few-shot configurations of the *random* prompts. The bold values are the highest accuracy for the row (K fixed).

The results from table 5 support our hypothesis. Now the models are almost incapable of making a single correct prediction. We also observe as expected that the largest model size outperforms the smaller models. It is the only model to consistently achieve above 0% accuracy across prompts sizes.

Overall there seems to be no benefit from increasing K. We suspect this is a results of the model sizes. If the models are too small for allowing complex generalization, then no amount of examples will make up for that. This intuition is also based on a few tests using the GPT-3 API, in which the 175B parameter model showed great prowess. That being said, when accuracy is this low, it proves difficult to differentiate between random fluctuations and clear patterns.

We also inspected the few correctly guessed samples, in order to better gauge what had aided the prediction. They all turned out to be very simple commands, having an average output length of 2.5. This again shows, that these smaller LLMs have a very weak model of the problem, only being able to interpret "walk and jump" and the likes.

The trend of performing better on shorter sequences also governs the BLEU prompts. In appendix L one can see that accuracy drops as target length increases. I would therefore expect the model accuracies across the entire test set to be significantly lower, but still clearly greater than that of the RNNs[9].

5 Conclusion

We are able to reproduce the findings of Lake and Baroni (2017) for experiment 1, 2 and 3 on almost all accounts. The models utilizing attentional mechanisms systematically underperformed the reported scores. We attribute this to the authors model selection, which lead to biased test scores. Following this, we formulated an in-context-learning equivalent to experiment 3. We observed far greater performance when comparing the smaller OPTs to the seq2seq models. However, it quickly showed, that this was not due to a deep understanding of the SCAN domain. This emphasizes, that the generalizational ability and systematic compoistionality of LLMs, are highly dependent on their scale and enormous parameter counts.

[8]See appendix K for an example prompt.

[9]One the subset of 1000 samples, the OPTs have already accumulated more correct predictions, then the RNNs did across all 7706 examples.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

N. Chomsky. 1957. Syntactic structures.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Diederik Kingma and Max Welling. 2014. Efficient gradient-based inference through transformations between bayes nets and neural nets. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1782–1790, Bejing, China. PMLR.

Brenden M. Lake and Marco Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation.

Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models.

6 Appendix

A SCAN Examples

jump	⇒	`JUMP`
jump left	⇒	`LTURN JUMP`
jump around right	⇒	`RTURN JUMP RTURN JUMP RTURN JUMP RTURN JUMP`
turn left twice	⇒	`LTURN LTURN`
jump thrice	⇒	`JUMP JUMP JUMP`
jump opposite left and walk thrice	⇒	`LTURN LTURN JUMP WALK WALK WALK`
jump opposite left after walk around left	⇒	`LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN LTURN JUMP`

Figure 6: Examples of SCAN commands (left) and the corresponding action sequences (right).

B Upscaled Charts

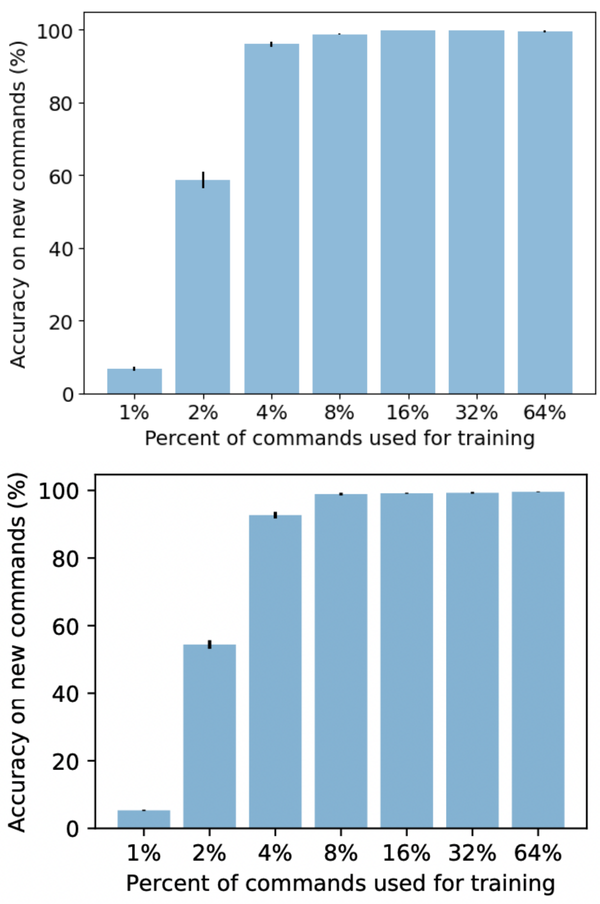


Figure 7: Comparison of reproduction (top) and reported (bottom) zero-shot generalization after training on a random subset of the SCAN tasks.

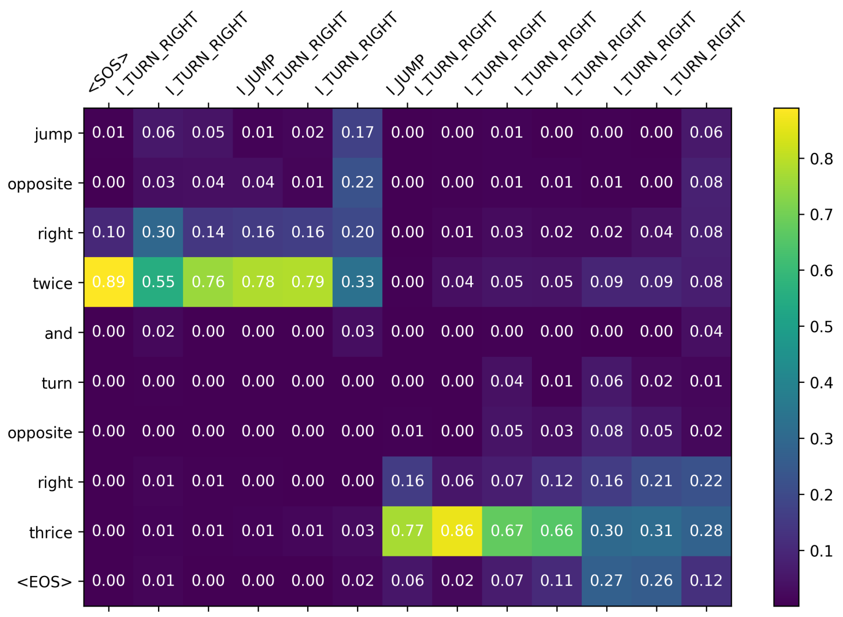


Figure 8: The attention map was produced using the best model architecture with the highest accuracy from experiment 2.

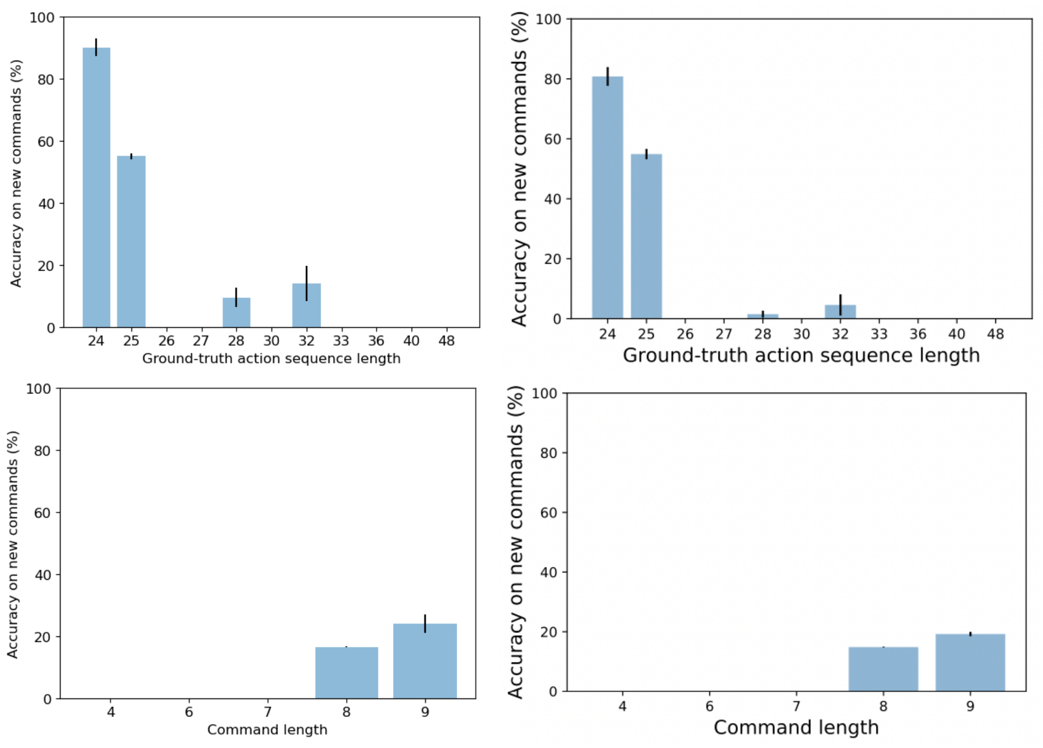


Figure 9: Top: accuracy distribution by action sequence length. Bottom: accuracy distribution by command length. Left: Reproduction. Right: Reported.

C Models

Model	**RNN Type**	**Layers**	**Hidden Units**	**Dropout**	**Attention**
Overall-Best	LSTM	2	200	0.5	False
Experiment 1 Best	LSTM	2	200	0	False
Experiment 2 Best	GRU	1	50	0.5	True
Experiment 3 Best `JUMP`	LSTM	1	100	0.1	True
Experiment 3 Best `LTURN`	GRU	1	100	0.1	True

Table 6: Model configurations

D Training Details

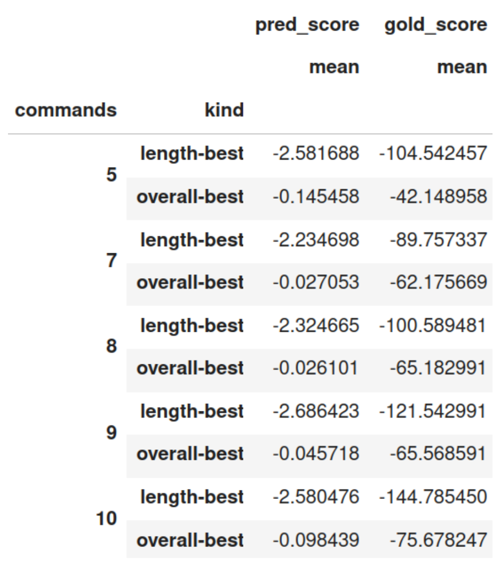
The paper does not explicitly define the batch size used for training. We therefore choose a batch size of 1 for all experiments in line with the aforementioned PyTorch tutorial. All experiments was trained using the Adam optimizer with a learning rate of 0.001 (Kingma and Welling, 2014). Negative log likelihood was used for the loss function. The models were shown 100.000 training examples with replacement. 50% of the examples would be generated in an autoregressive manor and 50% would utilize teacher forcing (Williams and Zipser, 1989). All gradients were clipped to a max norm of 5.

E Oracle at Evaluation

	Reimplementation	Reported
Overall	28.6 ± 7.8	23.6
Best	37.9 ± 11.2	60.2

Table 7: Findings for experiment 2 with oracle

F Search Related Errors



		pred_score	**gold_score**
		mean	**mean**
commands	**kind**		
5	**length-best**	-2.581688	-104.542457
	overall-best	-0.145458	-42.148958
7	**length-best**	-2.234698	-89.757337
	overall-best	-0.027053	-62.175669
8	**length-best**	-2.324665	-100.589481
	overall-best	-0.026101	-65.182991
9	**length-best**	-2.686423	-121.542991
	overall-best	-0.045718	-65.568591
10	**length-best**	-2.580476	-144.785450
	overall-best	-0.098439	-75.678247

Figure 10: Table showing mean log probabilities for the greedy output sequence and the golden (target) sequence.

G Nearest training commmands

In the two tables we see nearest training commands for representative commands, with the respective cosines. The paper was not very explicit about which model is shown in the cosine table. We interpreted it to be the best-overall model achieving median accuracy. Here, "jump" was trained in isolation while "run" was trained compositionally.

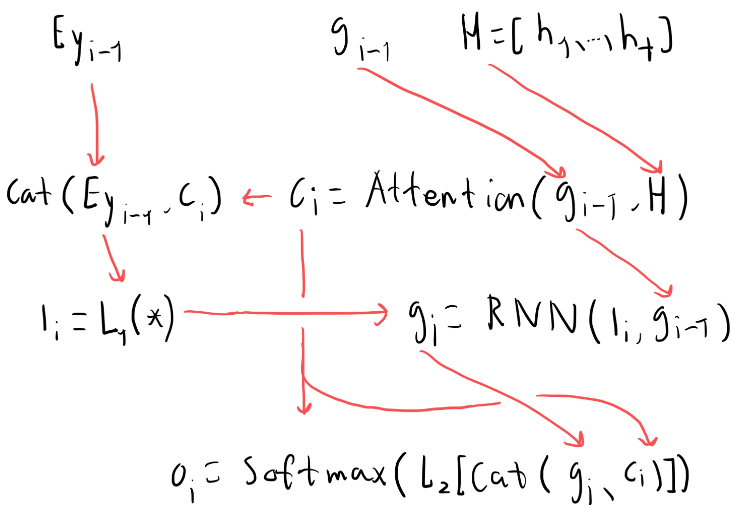
G.1 Reproduction

run		**jump**		**run twice**		**jump twice**	
walk	.67	walk	.24	walk twice	.86	run twice and run	.45
look	.66	run	.24	look twice	.86	look twice and run	.44
run after run	.52	look	.20	run twice and run twice	.71	walk twice and run	.44
walk after run	.52	turn right	.17	run thrice and run twice	.71	run twice and walk	.41
look after run	.52	turn left	.10	walk thrice and run twice	.69	walk twice and walk	.40

G.2 Reported

run		jump		run twice		jump twice	
look	.73	*run*	*.15*	look twice	.72	*walk and walk*	*.19*
walk	.65	*walk*	*.13*	run twice and look opposite right thrice	.65	*run and walk*	*.16*
walk after run	.55	*turn right*	*.12*	run twice and run right twice	.64	*walk opposite right and walk*	*.12*
run thrice after run	.50	*look right twice after walk twice*	*.09*	run twice and look opposite right twice	.63	*look right and walk*	*.12*
run twice after run	.49	*turn right after turn right*	*.09*	walk twice and run twice	.63	*walk right and walk*	*.11*

H Attention Diagram



Ey_{i-1}
g_{i-1}
$H=[h_1, \dots, h_T]$
$Cat(Ey_{i-1}, c_i)$
$c_i = Attention(g_{i-1}, H)$
$l_i = L_1(*)$
$g_i = RNN(l_i, g_{i-1})$
$o_i = Softmax(L_2[Cat(g_i, c_i)])$

Figure 11: Diagram over the Bahdanu Attention in the Decoder module

H.1 Terminology

g_{i-1} is the previous decoder hidden state.
$H = [h_1, \ldots, h_t]$ is a concatenation of the previous encoder hidden states up till time t.
L_1 and L_2 are learned linear transformations.
c_i is the context vector produced via the attention module.
RNN repræsents the recurrent neural network module. This may be an Elman RNN, LSTM or GRU.
$E_{y_{i-1}}$ is the embedding of the previously generated token.
The $attention$ module is implemented as described in Bahdanau et al. (2014), where the attention energies are calculated by $a\left(s_{i-1}, h_j\right) = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right)$. For more detail on this, we refer to the original paper.
cat is shorthand for the concatenation operation between two tensors.

I Grid Search

We quote from Lake and Baroni (2017): "Recurrent networks with attention have become increasingly popular in the last few years, and thus we also tested each network with and without an attentional mechanism ... A large-scale hyperparameter search was conducted that varied the number of layers (1 or 2), the number of hidden units per layer (25, 50, 100, 200, or 400), and the amount of dropout (0, 0.1, 0.5; applied to recurrent layers and word embeddings). Varying these hyperparameters leads to 180 different network architectures, all of which were run on each experiment and replicated 5 times each with different random initializations."

J Model results (25 runs)

[8.5%, 46.7%, 9.0%, 12.9%, 34.2%, 13.0%, 11.0%, 3.3%, 30.1%, 13.8%, 13.6%, 21.5%, 12.5%, 7.6%, 11.0%, 10.9%, 12.4%, 12.6%, 6.6%, 14.5%, 10.5%, 8.1%, 10.1%, 9.8%, 8.1%]

K Prompt Examples

Context ⇒	walk after look opposite right = `RTURN RTURN LOOK WALK` walk right and run thrice = `RTURN WALK RUN RUN RUN` turn right after walk around left thrice = `LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK LTURN WALK RTURN` jump = `JUMP` jump after run =
Target ⇒	`RUN JUMP`

Figure 12: Illustrative *random* prompt with $K = 4$

L Accuracy by target length

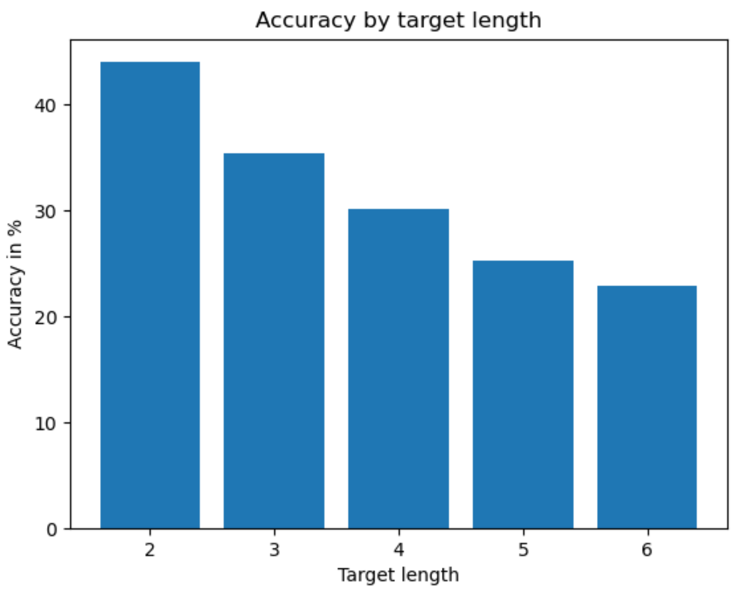


Figure 13: Accuracy across target lengths for OPT-2.7b with $K = 8$