Generating Demonstrations for In-Context Compositional Generalization in Grounded Language Learning

Sam Spilsbury
Dept. of Computer Science
Aalto University
Espoo, Finland

Pekka Marttinen
Dept. of Computer Science
Aalto University
Espoo, Finland
first.last@aalto.fi

Alexander Ilin
Dept. of Computer Science
Aalto University
Espoo, Finland

Abstract

In-Context-learning and few-shot prompting are viable methods compositional output generation. However, these methods can be very sensitive to the choice of support examples used. Retrieving good supports from the training data for a given test query is already a difficult problem, but in some cases solving this may not even be enough. We consider the setting of grounded language learning problems where finding relevant supports in the same or similar states as the query may be difficult. We design an agent which instead generates possible supports inputs and targets current state of the world, then uses them in-context-learning to solve the test query. We show substantially improved performance on a previously unsolved compositional generalization test without a loss of performance in other areas. The approach is general and can even scale to instructions expressed in natural language.

1 Introduction

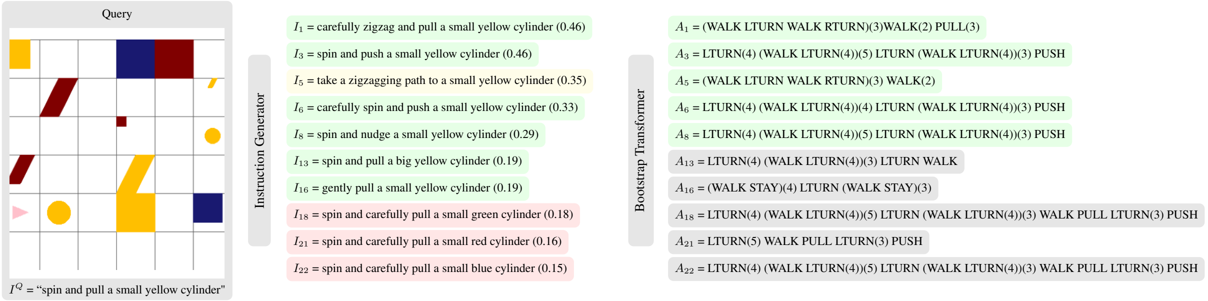
It is thought that a compositional understanding of language and the world (so-called *compositional generalization*) around is something that enables efficient learning in both humans (Chomsky, 1957; Tenenbaum, 2018) and machines (Sodhani et al., 2021; Jang et al., 2021). However, a long line of work and many different datasets show that Deep Learning approaches do not always achieve such compositional generalization. Some solutions to address this deficiency include modular architectures, data augmentation, and sparsity. A recent line of work concerns in-context learning (ICL). Instead of providing a query and asking for the target directly, a few examples of query-target pairs (*supports*) are also provided. Recent work indicates that supports covering the elements of the query can help enable compositional generalization even if neither shows the desired behaviour exactly (Gupta et al., 2023). A follow up question is how to find examples for each query. Most prior work suggests retrieval from the training data (Pasupat et al., 2021).

However, in the Grounded Language Learning case, retrieval approaches might not be sufficient to make compositional generalization by ICL work well. The expected outputs are conditional not only on the *query*, but also on the *state* of the world. Therefore, searching for nearby examples in the input space is problematic. Using the query alone means that it is unlikely that *state-relevant* examples will be retrieved. There might not be query-covering examples in the same state from the training data. Using *similar* states is also challenging because small changes in the state can result in large changes to the target sequence. Searching for nearby examples in the *output space* (Zemlyanskiy et al., 2022) is more promising, but it also relies on being able to find state-relevant covering outputs. It is difficult to make a retrieval-based strategy that works well in all cases.

Instead of retrieval, we suggest that generation of the supports based on the state might be a better alternative. We contribute the following:

- We confirm that in-context learning is a useful method for unlocking output compositional generalization in the grounded language learning context.
- We show that support selection for in-context learning is a crucial piece of the puzzle and that retrieval from the training set is not enough to get the best performance due to the challenge of the query state being potentially unobserved in the retrieval examples.
- We propose a new method, **DemoGen**, to generate the necessary supports which show different instructions and targets of which the query instruction requires a composition of. Our experiments on gSCAN, GSRR and ReaSCAN show that using in-context learning with these supports unlocks superior compositional generalization performance.
- We extend the gSCAN dataset to natural-language like instructions to show further that that **DemoGen** method can scale well to natural-language like instructions as well.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 15960–15991
November 12-16, 2024 ©2024 Association for Computational Linguistics



I^Q = “spin and pull a small yellow cylinder"

I_1 = carefully zigzag and pull a small yellow cylinder (0.46)
I_3 = spin and push a small yellow cylinder (0.46)
I_5 = take a zigzagging path to a small yellow cylinder (0.35)
I_6 = carefully spin and push a small yellow cylinder (0.33)
I_8 = spin and nudge a small yellow cylinder (0.29)
I_{13} = spin and pull a big yellow cylinder (0.19)
I_{16} = gently pull a small yellow cylinder (0.19)
I_{18} = spin and carefully pull a small green cylinder (0.18)
I_{21} = spin and carefully pull a small red cylinder (0.16)
I_{22} = spin and carefully pull a small blue cylinder (0.15)

A_1 = (WALK LTURN WALK RTURN)(3)WALK(2) PULL(3)
A_3 = LTURN(4) (WALK LTURN(4))(5) LTURN (WALK LTURN(4))(3) PUSH
A_5 = (WALK LTURN WALK RTURN)(3) WALK(2)
A_6 = LTURN(4) (WALK LTURN(4))(4) LTURN (WALK LTURN(4))(3) PUSH
A_8 = LTURN(4) (WALK LTURN(4))(5) LTURN (WALK LTURN(4))(3) PUSH
A_{13} = LTURN(4) (WALK LTURN(4))(3) LTURN WALK
A_{16} = (WALK STAY)(4) LTURN (WALK STAY)(3)
A_{18} = LTURN(4) (WALK LTURN(4))(5) LTURN (WALK LTURN(4))(3) WALK PULL LTURN(3) PUSH
A_{21} = LTURN(5) WALK PULL LTURN(3) PUSH
A_{22} = LTURN(4) (WALK LTURN(4))(5) LTURN (WALK LTURN(4))(3) WALK PULL LTURN(3) PUSH

Figure 1: Generating demonstrations on gSCAN with **DemoGen** for an ICL Transformer (Figure 2). The **Instruction Generator** takes as input the current state and I_q and produces similar instructions $I_1, ...I_n$ likely to occur in the same state, sorted by likelihood (parens). A **Bootstrap Transformer** trained on the training data generates the corresponding actions $A_1...A_n$ in that state. Some instructions are more helpful than others. Instructions in green, $I_{1,3,6,8,13,16}$ show both the correct object in I_q and also either one of the verb or adverb. Instructions in yellow, I_5 show the correct object, but an irrelevant verb and adverb combination. Instructions in red, $I_{18,21,22}$ show a different object to the target one. Actions in grey $A_{13,16,18,21,22}$ show an incorrect target sequence. As long as the instructions and actions in green are included in the support set, a sufficiently powerful model can use them and ignore the other supports.

2 Background

2.1 Compositional Generalization and Grounded Language Learning

Compositional Generalization refers to the ability of a system to learn the rule for how solutions to subproblems may be combined in some way, then apply the rule to unseen combinations of known sub-problem solutions. It can be seen in both the *inductive* and *productive* sense. In the *inductive* sense, the system must produce some known symbol in response to a unseen combination of known query inputs. In the *productive* sense, the system must produce some unseen combination of known output symbols. The capability of Deep Learning to perform compositional generalization has been studied extensively. Early experiments showed the challenge of doing so on both RNNs (Lake and Baroni, 2018) and Transformers (Hupkes et al., 2020) and many datasets have been created to demonstrate the problem, both with synthetic and “realistic" natural language data (Bastings et al., 2018; Kim and Linzen, 2020; Keysers et al., 2020; Li et al., 2021; Yin et al., 2021; Finegan-Dollak et al., 2018). As more datasets become available, so do approaches to handle the compositional generalization problem. Most approaches generally fall into some combination of data augmentation (Andreas, 2020; Li and McClelland, 2022; Chen et al., 2023; Qiu et al., 2022; Akyürek et al., 2021), neural module networks (Andreas et al., 2016b; Buch et al., 2021; D'Amario et al., 2021; Ruis and Lake, 2022) and meta-learning (Lake, 2019; Conklin et al., 2021).

The field of Grounded Language Learning is natural fit to study the problems of both inductive and productive compositional generalization. We can test inductive generalization by placing the agent in a state with a novel combination of input symbols. Productive generalization can be tested by giving instructions that require generating some novel combination of outputs conditioned on the state. While the former problem is extensively explored by related work, the latter has received less attention and therefore the focus of this work.

2.2 In-context Learning

Meta-learning and ICL are promising approaches for compositional generalization in sequence generation tasks. In this paradigm, a few *support inputs* and corresponding *support outputs* for a given *query* sequence are provided and the task is to predict the correct *target* sequence (Lake, 2019; Conklin et al., 2021). This has been popularized by the notion of ICL in large language models, where a few examples of the input-output pairs as well as a query are given as part of a *prompt*, then the target is predicted autoregressively (Brown et al., 2020; Min et al., 2022a), which has been shown to enable compositional generalization in sequence generation (Chen et al., 2022a; Logeswaran et al., 2020).

2.3 Support Selection for ICL

ICL methods are sensitive to the choice of support sets used. Mitchell et al. (2021) found that selecting supports that were not relevant to the task at hand degraded performance when using sequence based meta-learning with SCAN. As we also show in our experiments, ICL approachs with a poorly chosen procedure for selecting supports may be worse on all tasks compared to when no ICL is used at all.

Different approaches have been proposed for finding good examples. Many methods try to pick good examples from the training data, for example by using a similarity index (Pasupat et al., 2021), or with a metric that takes into account diversity and local structure coverage (Levy et al., 2022; Gupta et al., 2023; Ye et al., 2023). Such retrieval is potentially problematic, because getting relevant output supports requires that the

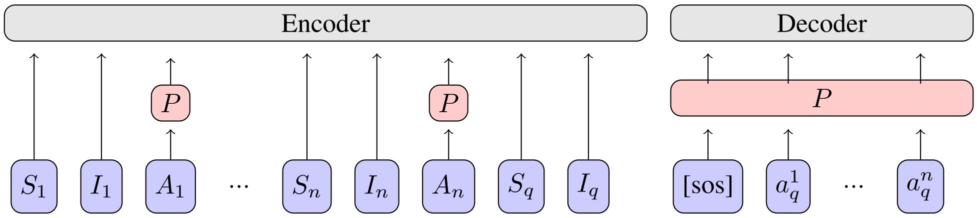


Figure 2: The model architecture for sequence-to-sequence ICL. Each support state $S_1, ..., S_n$, support instruction $I_1, ..., I_n$ and corresponding support targets $A_1, ..., A_n$, as well as the query state S_q and query instruction I_q are used as inputs to a Transformer Encoder (along with positional encoding). Right-shifted query targets $a_q^1, ..., a_q^n$ are used as inputs to a Transformer Decoder. Both the support targets and query targets use the same random permutation on every training step.

retrieved inputs are evaluated in the same or a very similar state, which can increase the complexity of the retrieval problem.

There are also generative approaches to create the support examples, for example subproblem decomposition (Yang et al., 2022), chain-of-thought (Wei et al., 2022), least-to-most-prompting (Zhou et al., 2023) asking for diverse examples (Yu et al., 2023). These approaches can get very impressive results on ungrounded compositional generalization benchmarks, but they have their own requirements including reliance on information in large language models or special helper prompts about the input structure. A hybrid of generation and retrieval is GandR Zemlyanskiy et al. (2022) which first guesses the output using a helper model and retrieves examples based on output similarity. Our work extends on the generated-example paradigm with the idea of *generating* support instructions for a query state, then *solving* those support instructions using a "bootstrap" model. We explain in Section 3.2 why this is important in the grounded language learning setting.

3 Method

In this section, we describe a method we call **DemoGen**. The method is designed to work with datasets where there is both an instruction and a state in the input.

3.1 In-Context Learning

ICL can be realized with a large-context encoder-decoder Transformer (see Figure 2). For an initial state S_q and instruction I_q, the model is trained to generate a sequence of targets $A = a_1^Q, ..., a_m^Q$ using a set of *support inputs* $I_1, ..., I_n$ and the corresponding *support outputs* $A_1, ..., A_n$.

The entire set of support states $S_1, ..., S_n$, support instructions $I_1, ..., I_n$ and corresponding support targets $A_1, ..., A_n$, along with the query state S_q and query instruction I_q are passed as one big sequence to the Transformer Encoder, using sine-cosine positional encoding in (Vaswani et al., 2017). Right-shifted query targets are used as inputs to the Transformer Decoder with causal masking.

We do not use a pre-trained model and train only on each problem's own training set to eliminate the risk of having pre-trained on the test set. To ensure that we still get in-context learning from the ICL Transformer, we use the technique of permuting the symbol-index mapping of the support and query targets on every training step (Chan et al., 2022).

3.2 Support Set Generation

Choosing the support inputs $I_1, ..., I_n$ and outputs $A_1, ..., A_n$ for the ICL model is not a trivial problem. **DemoGen** generates the support sets using two models trained on the training data - an **Instruction Generator** and **Bootstrap Transformer**.

Instruction Generator Support inputs are generated by a BART-like masked language model (Lewis et al., 2020). The model is trained to reconstruct a partially masked sentence. It is trained on a balanced dataset of all the instructions in the training data to ensure that inputs occurring less often have a reasonable chance of being sampled. To generate support inputs, some percentage of the tokens (including padding tokens) in the query I_q (in this work, 20%) are randomly masked and then the entire input is reconstructed by autoregressive decoding. This process is repeated $k \geq n$ times, to form $I_1, ..., I_k$. We deduplicate the samples and remove I_q from $I_1, ..., I_k$. We also *filter* the supports by the use of a *scoring model*. The scoring model estimates probability that a generated support is in-distribution, conditioned on any relevant context. The score is the length-normalized log-likelihood of generated support inputs. We assume that conditionally in-distribution supports are more likely to be solveable by the **Bootstrap Transformer** below. We take the top n by score to get $I_1, ..., I_n$.

Bootstrap Transformer Support outputs $A_1, ..., A_n$ are generated from the $(S_1, I_1), ..., (S_n, I_n)$ pairs by an Autoregressive Transformer model trained on the same training data. Examples of the generated support inputs and outputs are shown in Figure 1.

Generating both the support inputs and outputs has a few interesting advantages. Compared to retrieving on inputs, we can generate examples which we know will be relevant to the current state and also generate examples which might not be found in the training data for a given query. Compared to retrieving based on the predicted output, we can generate a greater diversity of supports which would be valid in the state, as opposed to fetching the same output over and over again in many different states. The only assumption we make is that the model used to generate the support targets is capable of inductive compositional generalization, but not necessarily productive compositional generalization. In practice, this is already true with the Transformer architecture (Qiu et al., 2021; Sikarwar et al., 2022). One challenge with generating the supports is that our support generator might come up with support inputs that are either not relevant or not solvable in the current state. We show in the experiments that the presence of irrelevant supports is not a problem as long as the other useful supports are also present.

4 Experiments

4.1 Dataset

We examine which dataset would be appropriate to evaluate DemoGen on. Since we know that in-context learning helps specifically when it comes to *productive* compositional generalization, we want the dataset to test this case. We also limit out dataset search to the state-conditioned setting, where it makes sense to generate demonstrations conditioned on the state. To really test our method, we also want a dataset using instructions in the form of natural language as well. We considered well-known compositional generalization and grounded language learning datasets. SCAN (Lake and Baroni, 2018), COGS (Kim and Linzen, 2020), SMCalFlow-CS (Yin et al., 2021) test productive generalization, but are not state-conditioned. RTFM (Zhong et al., 2020), ALFRED (Shridhar et al., 2020) and DescribeWorld (Weir et al., 2023) are state-conditioned but mainly test *inductive* generalization. MetaWorld (Yu et al., 2019) tests productive generalization, but in the few-shot learning context where examples are already given. gSCAN (Ruis et al., 2020) is the only dataset which tests *productive* generalization in a state-conditioned setting, however it uses very simplistic instructions. Based on this criteria, we choose to evaluate on gSCAN, but also extend it by rewriting the instructions using an LLM to resemble natural language, but we evaluate on ReaSCAN (Wu et al., 2021) and GSRR (Qiu et al., 2021) as well to confirm that our method works on instructions requiring more complex inductive compositional reasoning.

gSCAN, ReaSCAN and GSRR are Minigrid-based environment with a single training data set and 8 out-of-distribution test splits covering various compositional generalization scenarios. An agent receives an instruction with a target object, a verb to apply to that object and an adverb which affects both navigation and the verb. About 360,000 demonstrations of navigating to various objects and performing some task on them with various adverbs are provided as a training set. A *success* happens when the agent performs the expected sequence of actions exactly. The input and action vocabularies are small and the instructions constructed using a simple grammar. Typically the instructions follow the form "[verb] a [size] [color] [object] [adverb]", where [size], [color] and [adverb] are sometimes omitted. The in-distribution split is 100% solvable by deep learning.

More challenging are the eight out-of-distribution test splits. Splits B, C, E, F in gSCAN require *inductive* generalization, for example identifying a "red square" as a goal in split C and a size-3 object being "small" in relation to other objects in split E. The extensions GSRR and ReaSCAN test further such scenarios, for example by specifying the target object as one that is relative to some other object, requiring a few hops of reasoning. Further description of each test scenario is given in the appendix. Splits D, G and H of gSCAN require *productive* generalization at testing-time. Split D requires navigating to an object that is south-west of the agent, which in practice requires the production of `LTURN(2) ... LTURN`[1]. Split H requires composing a the verb "pull" with the adverb "while spinning", which requires the production of novel fragments `LTURN(4) PULL`. Split G is a few-shot learning split for a new behaviour "cautiously".

	Parses	Words	Zipf α	RMSE
gSCAN	18	18	1.99	0.11
GSRR	234	20	1.90	0.10
ReaSCAN	1400	35	1.26	0.04
NL-gSCAN	1550	859	1.29	0.01

Table 1: Linguistic properties of the baseline (gSCAN), extensions (GSRR and ReaSCAN) and paraphrased datasets (NL-gSCAN)

.

Natural Language Instructions We also extend the gSCAN dataset such that the instructions are less formulaic and more like natural language. By prompting the `openai-gpt3.5` model with 25 different examples of paraphrases for an instruction, we can generate paraphrases of all the other instructions in the dataset. To validate that the paraphrased dataset looks more like natural language, we estimate the α parameter (closer to 1.0 meaning more natural) for a Zipf distribution using maximum likelihood estimation using the method in (Clauset et al., 2009) and also calculate the number of unique parses with spaCy. The paraphrased data has an α of 1.29 vs 1.99 along with a better fit, and there is a greater diversity of both words (18 vs 859) and syntax parses (18 vs 1550). The target object description was retained in approximately 99% of cases. Examples of

[1] In this work, where an action or subsequence is repeated n times, we use the notation `(ACT1 ACT2)(n)`

paraphrases and further analysis given in Appendix L. Paraphrased instructions are also shown in Figure 1.

Related Work on gSCAN Various approaches to gSCAN including graph networks (Gao et al., 2020), linguistic-assisted attention (Kuo et al., 2021), symbolic reasoning (Nye et al., 2021), auxiliary tasks (Jiang and Bansal, 2021; Hein and Diepold, 2022), modular networks (Heinze-Deml and Bouchacourt, 2020; Ruis and Lake, 2022), logic programming (Yang et al., 2023) and data augmentation (Setzler et al., 2022; Ruis and Lake, 2022) have been proposed. These approaches tend to make some trade-off between performance and generalizability. Transformers have been shown to work well on on the *inductive* category of splits (Qiu et al., 2021) as well as on ReaSCAN and GSRR (Sikarwar et al., 2022), but there is no general approach which works well on the *productive* category. In this work, we aim to show that an ICL approach along with a support generation strategy that does not assume too much about the problem is a feasible general approach for the problems we study.

4.2 What makes for good supports?

We first explore what sort of supports work well for gSCAN. These methods are based on pre-existing knowledge of the dataset. When we perform experiments with the ICL Transformer, we use the architecture described in Section 3.1 trained to 300,000 steps with batch size 128, hidden size of 512, 8 attention heads, 12 layers and 16 supports per query example. Training was run for 300,000 iterations over 10 seeds. We perform evaluation every 5000 steps on a random subsample of the validation data and the best by split-A (in-distribution) performance are reported. Detailed information on hyperparmeters is given in Appendix C

Heuristic Select the best instructions and outputs for a given state which show; 1) going to the same object, 2) showing the target verb in combination with other adverbs, 3) showing the target adverb in combination with other verbs. Note that the generated supports might contain test-set input-output pairs, meaning that we assume extra knowledge not available to the learning agent. The heuristic can be seen as an upper bound on we could expect from an optimal demonstration generator.

Random Instructions (RD) Support instructions are selected randomly, without the use of the heuristic described above. Instructions can be about any object in the same state, not just the target one.

Other States (OS) We generate the same instructions as in the Heuristic approach but demonstrations are in states different to the query state. Such states are extracted from the training data. The sampled states are also included in the supports and used by the ICL Transformer. If the training data does not contain a state with the same instruction as the one generated by the expert, that instruction is not included in the support set. Table 3 shows the coverage of the supports over the query according to some hand-written metrics. Heuristic gets full coverage in every category. If we demonstrate random instructions in the same state (RD), only show demonstrations describing the same object 16% of the time (1). If we pick known good instructions for the query demonstrated in different states (OS) then we often describe the correct object, but the outputs look very different to the query, because the starts (2) or finishes (3) in a different position and the agent-target distance is often different (4). This is also reflected in the ICL Transformer performance in Table 2, where demonstrations of relevant instructions in different states show a very wide performance gap and demonstrations in the same state with randomly chosen instructions perform better, but still overall worse than the Heuristic. This supports the idea that our support selection procedure should find demonstrations that both cover the query input and also do so in the same state as the query.

4.3 Retrieval vs Generation

In the real world, we don't have access to a heuristic function to generate good supports. Instead we have to come up with them using the data we are already given. We can either try to retrieve good supports from the dataset or try to generate them somehow. We compare the following state-of-the-art retrieval methods tested on other productive compositional generalization problems and compare them to DemoGen. Further details of implementations are given in Appendix E and D

Coverage Retrieval (CR, CovR) Supports are retrieved using a similarity index on states and instructions, then chosen based on query coverage similar to Gupta et al. (2023). Instructions are encoded with `sentence-transformers` and states are flattened, one-hot encoded, then projected along their 320 principal components. The influence of the state and instructions on encoding similarity is balanced by multiplying instruction vectors by the ratio of the state vector norm to the instruction vector norm, contatenating and renormalizing. For each query input and state, we find the 128 nearest neighbours, then rank them descending by their one and two-gram coverage. Examples from the retrievals are chosen greedily such that all the one-grams and two-grams in the query are covered maximally.

GandR (GR) Supports are retrieved using the Generate-and-Retrieve strategy (Zemlyanskiy et al., 2022). In that strategy, a “helper" model trained on the training data makes an initial guess for the outputs of a given query in a state, even if that query is out of distribution. Then examples for later in-context learning are fetched by similarity of their output sequence to the guessed output sequence. In our implementation, similar to CovR, both query instructions and *outputs* are vector encoded and retrieved from a similarity index. 128 examples are chosen and we greedily pick examples from the 128 nearest output neighbours covering the

		No ICL		Algorithmic			Retrieval		Generation		
		TF	FT	Heuristic	RD	OS	CovR	GandR	DemoGen	DG-NP	DG-NF
gSCAN	A	**1.0**	1.0	1.0	0.77	0.99	0.99 ± .01	0.99 ± .01	1.0 ± .01	0.94 ± .06	0.96 ± .02
	B	**1.0**	1.0	1.0	0.62	0.0	0.98 ± .01	0.88 ± .05	1.0 ± .01	0.92 ± .05	0.92 ± .02
	C	0.96	1.0	1.0	0.66	0.2	0.83 ± .30	0.92 ± .03	**0.98 ± .02**	0.72 ± .27	0.85 ± .03
	D	0.01	0.16	0.50	0.0	0.0	0.0 ± .00	0.0 ± .00	**0.03 ± .02**	0.0 ± .00	0.01 ± .01
	E	**1.0**	1.0	1.0	0.59	0.0	0.99 ± .01	0.99 ± .01	0.99 ± .01	0.92 ± .09	0.86 ± .03
	F	**1.0**	1.0	1.0	0.75	0.99	0.99 ± .01	0.99 ± .01	0.99 ± .01	0.92 ± .08	0.95 ± .01
	G	0.0	0.0	0.0	0.0	0.0	0.0 ± .00	0.0 ± .00	0.0 ± .00	0.0 ± .00	0.0 ± .00
	H	0.22	0.22	0.86	0.15	0.0	0.56 ± .10	0.17 ± .01	**0.8 ± .05**	0.18 ± .02	0.62 ± .2
ReaSCAN	A1	0.99	0.99	-	-	-	0.89 ± .03	0.86 ± .03	0.91 ± .04	0.94 ± .01	**0.97 ± .00**
	A2	0.92	0.93	-	-	-	0.77 ± .07	0.95 ± .01	0.89 ± .05	0.87 ± .01	**0.96 ± .00**
	B1	0.94	0.78	-	-	-	0.88 ± .03	0.95 ± .03	0.85 ± .04	0.81 ± .01	**0.96 ± .00**
	B2	0.88	0.51	-	-	-	0.89 ± .03	0.90 ± .01	0.81 ± .07	0.8 ± .01	**0.92 ± .01**
	C1	**0.67**	0.19	-	-	-	0.32 ± .02	0.25 ± .01	0.28 ± .02	0.28 ± .01	0.25 ± .01
	C2	0.19	0.19	-	-	-	0.55 ± .05	0.62 ± .04	0.66 ± .02	**0.71 ± .00**	0.65 ± .03
SR	I	1.0	1.0	-	-	-	1.0 ± .00	0.99 ± .00	0.99 ± .01	0.99 ± .00	0.93 ± .13
	II	0.99	0.97	-	-	-	0.99 ± .00	0.99 ± .00	0.98 ± .00	0.99 ± .01	0.92 ± .13
	III	0.99	1.0	-	-	-	0.99 ± .00	0.98 ± .00	0.98 ± .00	0.98 ± .01	0.91 ± .14
	IV	1.0	1.0	-	-	-	0.99 ± .00	0.99 ± .00	0.98 ± .00	0.99 ± .00	0.93 ± .13
	V	0.82	0.77	-	-	-	**0.97 ± .00**	0.88 ± .00	0.93 ± .01	0.86 ± .07	0.82 ± .13
	VI	0.81	0.80	-	-	-	0.9 ± .00	0.88 ± .01	**0.90 ± .13**	0.69 ± .18	0.83 ± .13

Table 2: Success rates on reference datasets for different splits. Numbers are ± standard deviation over 10 seeds, measured after 300,000 steps. Variances are shown only for retrieval and generation experiments and are negligible on other experiments. Algorithmic, Retrieval and Generation all use ICL Transformer as the architecture, with supports generated by each method. TF is a Transformer baseline and FT is the same Transformer fine-tuned on generated demonstrations from DemoGen. Best non-oracle results bolded.

	RD	OS	CR	GR	DG
(1) Desc. Obj.	0.16	1.00	0.33	0.68	0.33
(2) Agent Pos.	1.00	0.03	1.00	0.08	1.00
(3) Tgt. Pos.	0.16	0.03	0.39	0.08	0.44
(4) Same Diff.	0.16	0.02	0.39	0.09	0.44
(5) Tgt. Obj.	0.16	0.19	0.27	0.14	0.44
(6) Verb & (1)	0.16	0.43	0.88	0.15	1.00
(7) Advb & (1)	0.16	0.33	0.78	0.51	0.88
(8) (6) & (7)	0.16	0.19	0.70	0.00	0.88
(9) (4) & (8)	0.16	0.00	0.62	0.00	0.88

Table 3: Fraction of supports matching criteria from on each generation method on Split H. Omitted is **Heuristic**, which is 1.0 in every category. (6)-(8) are calculated based on whether any support in a query's supports match that criteria. Other splits are shown in Appendix F

query input to avoid picking the same (non-covering) instruction many times.

DemoGen (DG) Our generation strategy as described in Section 3.2. 2048 instructions are sampled from the language model, deduplicated, and ranked to get the top 16 instructions and corresponding support targets for the query state. A Transformer with the same architecture as given in Section 3.1 is used as the Bootstrap model.

4.4 Retrieval Methods vs Generation

The main challenge for retrieval methods is that the supports inputs and outputs for some test splits may not exist in the training data. In gSCAN, we also found that most states don't have close near neighbours. An average example's nearest neighbour had a hamming similarity of 0.74 ± .107 (i.e., 10 of 36 cells would be different in the nearest neighbour). Detailed similarity analysis is made in Appendix A.1. This is also reflected in the properties of retrieved supports. In Table 3, the distance between the agent and the target object is often different in the query versus the supports (4) and there are fewer demonstrations showing the same exact same target object (5). They also do not always have both (8) the correct verb (6) and adverb (7) in the retrieved supports. On GandR the adverb can significantly change the outputs, so supports with the same verb (but without the adverb) are not selected. For both methods there are even fewer cases where there is least one demonstration of both the correct verb and of the adverb on on the same path (9).

Deficiencies in query coverage aside, these baselines are still stronger on Split H than many previously published results. **CovR** retrieves examples that are very close to the query state like and gets a success rate of 56% on gSCAN Split H and 44% on NL-gSCAN Split H with high variance, However on Split C, CovR loses performance compared to the baseline and has high

	TF	CovR	GandR	DemoG
A	**1.0 ± .00**	0.98 ± .03	0.94 ± .01	0.99 ± .00
B	**0.99 ± .00**	0.93 ± .08	0.92 ± .06	0.96 ± .00
C	**0.99 ± .03**	0.68 ± .37	0.9 ± .04	0.97 ± .00
D	**0.08 ± .16**	0.0 ± .00	0.0 ± .00	0.01 ± .01
E	0.98 ± .03	0.95 ± .08	0.89 ± .01	0.98 ± .00
F	**1.0 ± .00**	0.88 ± .11	0.94 ± .02	0.98 ± .00
G	0.0 ± .00	0.0 ± .00	0.0 ± .00	0.0 ± .00
H	0.19 ± .03	0.44 ± .23	0.17 ± .00	**0.59 ± .06**

Table 4: Success rates for a non-ICL Transformer (TF) retrieval baselines and DemoGen on NL-gSCAN. Best results bolded.

variance between seeds on both datasets. The other inductive generalization splits on NL-gSCAN also have small but not negligible loss compared to a non-ICL Transformer when using CovR to retrieve the supports. For ReaSCAN and GSRR retrieval performance is also competitive and is actually a bit closer to what we get with support generation, possibly because the supports are more similar to the query (as discussed in Appendix A.1). **GandR** gets 17% on gSCAN Split H, but retains good performance on the other splits. However it loses about 10 points on gSCAN splits B and C and 5 points on Split F of NL-gSCAN compared to the baseline.

Generating the Supports How does generating the supports with **DemoGen** compare? In Table 3 we see that the generated instructions cover the different aspects of the instruction and they are made in the same state. This means that the agent starting position is preserved (2), the path between the starting the target position (between supports and target) is better preserved (4) and, crucially, both the correct verb (6) and adverb (7) are present in the demonstration in combination with the correct object. Demonstrating the right things also has an impact on performance. **DemoGen**, gets 80% on productive generalization Split H for gSCAN and even 59% for the more challenging NL-gSCAN. Performance also remains good on the the inductive generalization splits for both datasets. We provide a summary and detailed comparison to prior work on gSCAN in Appendix B. Aside from (Hein and Diepold, 2022), a specialized architecture with some additional supervision, ours is the best result on Split H. On ReaSCAN and GSRR, ICL can get very strong performance on the challenging C2 and VI splits (beating the state of the art in Sikarwar et al. (2022) for C2) and is competitive on other splits. We also show that DemoGen generated supports gets reasonable performance in other in-context learning setups, for example with an image-based gSCAN dataset in Appendix N and also when using text-encoded states with LLaMA 3 in Appendix O.

On Splits D and G, performance on retrieval methods and DemoGen is still not good. The reason is they require generation of a pattern that won't be seen in the outputs in any permutation of the labels. In the case of Split D, it requires `LTURN(2) WALK(n) LTURN(1) WALK(n)`. Only 6% of the data matches this pattern in any index-label permutation. In the case of split G, `(LTURN RTURN(3) LTURN WALK)(n)` is required. Only 0.0001% of training data matches that up to a permutation. In contrast, Split H requires `(LTURN(4) PULL(n))`, and there are many examples from the “push a [size] [color] [object]“ set of instructions matching that up to a permutation.

Comparing retrieval and generation, we see that retrieval is a good start for finding good supports, and they can still get close to selecting supports with a good heuristic in the state-conditioned setting, but generating the supports usually outperforms retrieval, especially in the productive setting.

4.5 Ablations and Further Questions

	Valid	Correct	C & V	C | V
A	0.79	0.70	0.70	0.88
B	0.73	0.64	0.64	0.88
C	0.61	0.50	0.50	0.83
D	0.65	0.24	0.24	0.36
E	0.78	0.66	0.66	0.84
F	0.73	0.63	0.63	0.87
G	0.79	0.72	0.72	0.91
H	0.79	0.56	0.56	0.71

Table 5: DemoGen supports: Fraction of valid instructions, correct targets, correct and valid (C & V) and correct given valid (C | V) on synthetic data by split, according to an oracle function.

Quality of Supports Ideally, support should comprise *valid* support inputs (eg, tasks that are actually solvable in a state) and they should be *correct* enough to facilitate ICL. We investigated this on supports generated by our method and reported the results in Table 5. On average, about 77% of generated support inputs are valid. A support output is *correct* if it matches what an oracle generator generated for the corresponding instruction and state. 50% of the support pairs were both correct and valid. The number is clearly lower on splits where a Transformer is not able to solve the task well. For example on Split H, there may be “pull an [object] while spinning" in the support inputs, where [object] is not the target object.

Criteria	Success Rate	Change
Remove tgt obj.	0.67 ± .17	0.13
Remove adverb	0.3 ± .16	0.5
Remove verb	0.21 ± .04	0.59

Table 6: DemoGen Split H success rate with 16 supports, but excluding specified supports.

Effect of Supports We also examine how important it is to have the right demonstrations at inference time by removing demonstrations matching certain criteria from the support set. Removing those any containing the same object makes a 13 point impact on success rate. Bigger changes come from removing those any having the same adverb (50 points) or verb (59 points). Learning with permutations alone is not enough - its also important that the supports cover the types of output behaviour that are found in the target. We found that on Split H there is a correlation between the exact match performance of examples and the mean similarity (r=0.21) of the supports to the query. Diversity within the supports is negatively correlated (r=-0.21), but a closer examination in Appendix P shows that there is an inflection point where increased sample diversity boosts performance, then eventually decreases it (because sample diversity is strongly negatively correlated with sample query relevance at r=-0.91). Finally, the number of demonstrations also matters - with 4 demonstrations and less, exact match performance on all splits reduces to about 40%, and the best performance is found with around 12 demonstrations, where the results are close to the reported ones.

Permutations Our ICL Transformer uses a different symbol-index mapping on each training step. On gSCAN, the sequence "`WALK(5) RTURN WALK(5)`" would be translated into "`RTURN(5) LTURN RTURN(5)`" under the permutation `WALK` → `RTURN`, `RTURN` → `LTURN`. One concern is the possibility that a *query target* with the same symbols for `pull` ... `while spinning` is generated after permutation during training, however the probability of this happening is very low. We measured that for a single pass through the training data, approximately 3% of the generated support instructions matched `pull` ... `while spinning`, 0.3% of the permuted query outputs matched `PULL` actions followed by four `LTURN` instructions, and their intersection was 0.001% of all sampled supports.

Architectural Ablations We also compare the effect of various ablations on gSCAN success rate in Table 2. Fine-Tuning (FT) on the supports generated by DemoGen improves performance marginally on Split D, but not Split H, which shows the importance of using in-context learning for productive generalization. Removing the permuter block (DG-NP) reduces performance to a similar level of not using ICL at all, though it does marginally improve performance for the inductive split C2 on ReaSCAN. Remvoing Filtering (DG-NF) reduces average split C and split H performance by about 13 and 20 points respectively with higher variance. We also tried other variants of the Transformer architecture, including RoFormer (Su et al., 2024), Universal Transformer (Dehghani et al., 2019) and Perceiver (Jaegle et al., 2022), which all had similar results compared to a regular Transformer.

5 Conclusion

In-Context Learning can help improve performance on challenging compositional generalization problems, but the choice of support examples is crucial to its performance. In the grounded-language learning case, retrieval may not be enough to get good supports. Generated supports better cover what is required for productive generalization as shown in our support analysis and ablation studies.

We proposed **DemoGen**, a method for sampling support inputs from an autoregressive language model conditioned on the query state, then and solving them using a bootstrap model. When **DemoGen** used with in-context learning, our method outperforms both the best general non-retrieval architectures and also other strong retrieval based baselines on the challenging Split H of gSCAN, while retaining good performance on other datasets. Our method is general and also works well even if the instructions resemble natural language.

6 Limitations

In this section, we discuss the limitations of our work.

gSCAN, GSRR and ReaSCAN are synthetic and with initially simple instructions that eventually become more complex. We wanted to evaluate on instructions that were challenging like natural language, but we did not have the resources to crowdsource annotations for every data point in gSCAN. Therefore, we relied on commercial large language models to generate similar instructions instead. These instructions aren't a substitute for exactly human-generated language, but they are a good approximation.

In this work we decided to dive deep into evaluation on gSCAN and their derivatives. We are not aware of any other datasets which test the output-sequence level compositional behaviour generalization demanded by for example gSCAN Split H. The second reason is that gSCAN is a diagnostic dataset with output sequence rules which are not noisy and easy to understand for humans. This means that we can more precisely measure the properties of the generated supports and their effectiveness with respect to performance on the problem. Evaluating on the other gSCAN derivatives is still a limitation, but it does show that the method can generalize to quite demanding instructions, even if those instructions are noisy.

Another limitation of this work is that supports need to be generated at test time for the test set. In this work, we pre-generated the supports for the test set, though a real-time application of this work on unseen examples would need to also run the generation process. It currently takes about five seconds per batch of 128 inference queries to generate all the relevant demonstrations, sort them and run the inference using generated demonstrations in the context window. There are also other methods to improve the performance of the support input and support output procedure, for example quantization (Dettmers et al., 2022), KV-caching (Pope et al., 2022) and speculative decoding (Leviathan et al., 2023) etc.

7 Ethics

We used commercial large language models to generate paraphrases of the inputs to test the scalability of our method to natural language data in Section 4.1. These commercial large language models come with their own range of documented ethical issues, such as the capability to amplify harmful biases and misinformation, labour exploitation in training, energy consumption and permission to use web-scale training data. There is also an economic ethical aspect, where the use of the large language model displaces humans who may have been willing to perform the labelling. For our usecase, it was by many orders of magnitude cheaper to use the large language model than crowd-sourced labelling at a fair wage. On the other hand, we believe that there are better uses of human time than paraphrasing hundreds of thousands of examples of simple navigation problems for the purpose of producing a single research paper.

Our work covers the foundational issue of compositional generalization in grounded language learning, so we don't expect direct applications of it to have the potential to cause social harm. However, the work should be adapted with care. In particular, it is important that the model generating the supports for ICL is actually generating supports which are useful for generating the downstream problem. Generating outputs to a problem with generated wrong input-output pairs is likely to result in even more wrong outputs. Our work shouldn't be deployed in safety critical situations, but instead should be seen as a step towards achieving better data-driven compositional generalization.

8 Code and Resources

Our project code can be found at `https://github.com/aalto-ai/demogen`.

The paraphrased gSCAN dataset referred to in Section 4.1 can be found at `https://emnlp-2024-demogen-submission.s3.eu-north-1.amazonaws.com/dataset-paraphrased.txt`.

9 Computational Resource Usage and Reproducibility Requirements

Experiments were run on our internal GPU cluster. Running a ICL experiment to 300,000 iterations takes about 3 days on a MI250x GPU. For 6 different experiment runs with 10 seeds each, the total compute time is about 330 GPU-days, though the experiments can be run in parallel. The number of GPU-days we used to produce this work was much higher, because of tweaks to the experimental conditions, debugging, restarting failed jobs, etc.

Acknowledgements

We would like to acknowledge the anonymous reviewers of this paper in its various submissions, as well as our colleagues Nicola Dainese and Ananth Mahadevan for their valuable feedback on prior versions of this work. Computational resources were generously provided by the Aalto Science-IT project and CSC – IT Center for Science, Finland. We also acknowledge the the support within the Academy of Finland Flagship programme: Finnish Center for Artificial Intelligence (FCAI).

References

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2021. Learning to recombine and resample data for compositional generalization. In *International Conference on Learning Representations*.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Learning to compose neural networks for question answering. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Neural module networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jasmijn Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: SCAN both left and right. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*.

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. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Shyamal Buch, Li Fei-Fei, and Noah D. Goodman. 2021. Neural event semantics for grounded language understanding. *Transactions of the Association for Computational Linguistics*, 9.

Stephanie Chan, Adam Santoro, Andrew K. Lampinen, Jane Wang, Aaditya Singh, Pierre H. Richemond, James L. McClelland, and Felix Hill. 2022. Data distributional properties drive emergent in-context learning in transformers. In *NeurIPS*.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. 2018. Gated-attention architectures for task-oriented language grounding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*.

Wei-Lin Chen, Cheng-Kuang Wu, Yun-Nung Chen, and Hsin-Hsi Chen. 2023. Self-icl: Zero-shot in-context learning with self-generated demonstrations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 15651–15662. Association for Computational Linguistics.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022a. Meta-learning via language model in-context tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*.

Zining Chen, Weiqiu Wang, Zhicheng Zhao, Aidong Men, and Hong Chen. 2022b. Bag of tricks for out-of-distribution generalization. In *Computer Vision - ECCV 2022 Workshops - Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part VI*, volume 13806 of *Lecture Notes in Computer Science*, pages 465–476. Springer.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. BabyAI: A platform to study the sample efficiency of grounded language learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Noam Chomsky. 1957. *Syntactic Structures*.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. 2009. Power-law distributions in empirical data. *SIAM Rev.*, 51(4).

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*.

Vanessa D'Amario, Tomotake Sasaki, and Xavier Boix. 2021. How modular should neural module networks be for systematic generalization? In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates*, volume abs/2205.12548.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2023. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*.

Tong Gao, Qi Huang, and Raymond J. Mooney. 2020. Systematic generalization on gSCAN with language conditioned embedding. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020*.

Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. 2022. LISA: Learning interpretable skill abstractions from language. In *Advances in Neural Information Processing Systems*.

Prasoon Goyal, Raymond J. Mooney, and Scott Niekum. 2021. Zero-shot task adaptation using natural language. *arXiv:2106.02972*.

Shivanshu Gupta, Matt Gardner, and Sameer Singh. 2023. Coverage-based example selection for in-context learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 13924–13950. Association for Computational Linguistics.

Alice Hein and Klaus Diepold. 2022. A minimal model for compositional generalization on gscan. In *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2022, Abu Dhabi, United Arab Emirates (Hybrid), December 8, 2022*.

Christina Heinze-Deml and Diane Bouchacourt. 2020. Think before you act: A simple baseline for compositional generalization. *arXiv:2009.13962*, 2009.13962.

Felix Hill, Andrew K. Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L. McClelland, and Adam Santoro. 2020. Environmental drivers of systematicity and generalization in a situated agent. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *J. Artif. Intell. Res.*, 67.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. 2022. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2021. BC-Z: zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning, 8-11 November 2021, London, UK*.

Yichen Jiang and Mohit Bansal. 2021. Inducing transformer's compositional generalization ability via auxiliary sequence prediction tasks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the Thirty-Sixth Conference on Neural Information Processing Systems*, volume 35.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2021. Compositional networks enable systematic generalization for grounded language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*.

Brenden M. Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing*

Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80.

Brenden M. Lake, Tal Linzen, and Marco Baroni. 2019. Human few-shot learning of compositional instructions. In *Proceedings of the 41th Annual Meeting of the Cognitive Science Society, CogSci 2019: Creativity + Cognition + Computation, Montreal, Canada, July 24-27, 2019*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *ICML*, pages 19274–19286.

Itay Levy, Ben Bogin, and Jonathan Berant. 2022. Diverse demonstrations improve in-context compositional generalization. In *61st Annual Meeting of the Association of Computational Linguistics*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*.

Yafu Li, Yongjing Yin, Yulong Chen, and Yue Zhang. 2021. On compositional generalization of neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*.

Yuxuan Li and James McClelland. 2022. Systematic generalization and emergent structures in transformers trained on structured tasks. In *NeurIPS '22 Workshop on All Things Attention: Bridging Different Perspectives on Attention*.

Lajanugen Logeswaran, Yao Fu, Moontae Lee, and Honglak Lee. 2022. Few-shot subgoal planning with language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*.

Lajanugen Logeswaran, Ann Lee, Myle Ott, Honglak Lee, Marc'Aurelio Ranzato, and Arthur Szlam. 2020. Few-shot sequence learning with transformers. In *Workshop on Meta-Learning, NeurIPS 2020, virtual*, volume abs/2012.09543.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022a. Metaicl: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*.

So Yeon Min, Devendra Singh Chaplot, Pradeep Kumar Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2022b. FILM: Following instructions in language with modular methods. In *International Conference on Learning Representations*.

Eric Mitchell, Chelsea Finn, and Christopher D. Manning. 2021. Challenges of acquiring compositional inductive biases via meta-learning. In *AAAI Workshop on Meta-Learning and MetaDL Challenge, MetaDL@AAAI 2021, virtual, February 9, 2021*, volume 140.

Maxwell Nye, Michael Henry Tessler, Joshua B. Tenenbaum, and Brenden M. Lake. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In *Advances in Neural Information Processing Systems*.

Panupong Pasupat, Yuan Zhang, and Kelvin Guu. 2021. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently scaling transformer inference. *CoRR*, abs/2211.05102.

Linlu Qiu, Hexiang Hu, Bowen Zhang, Peter Shaw, and Fei Sha. 2021. Systematic generalization on gSCAN: What is nearly solved and what is next? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. 2020. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Laura Ruis and Brenden M. Lake. 2022. Improving systematic generalization through modularity and augmentation. In *Proceedings of the 44th Annual Conference of the Cognitive Science Society*.

Matthew Setzler, Scott Howland, and Lauren A. Phillips. 2022. Recursive decoding: A situated cognition approach to compositional generation in grounded language understanding. *Arxiv:2201.11766*.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10737–10746. Computer Vision Foundation / IEEE.

Ankur Sikarwar, Arkil Patel, and Navin Goyal. 2022. When can transformers ground and compose: Insights from compositional generalization benchmarks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates*.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. 2021. Multi-task reinforcement learning with context-based representations. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Josh Tenenbaum. 2018. Building machines that learn and think like people. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Nathaniel Weir, Xingdi Yuan, Marc-Alexandre Côté, Matthew Hausknecht, Romain Laroche, Ida Momennejad, Harm Van Seijen, and Benjamin Van Durme. 2023. One-shot learning from a demonstration with hierarchical latent language. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 2388–2390, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Zhengxuan Wu, Elisa Kreiss, Desmond Ong, and Christopher Potts. 2021. ReaSCAN: Compositional reasoning in language grounding. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. 2022. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. In *Findings of the Association for Computational Linguistics: NAACL 2022*.

Zhang Yang, Adam Ishay, and Joohynung Lee. 2023. Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the 61th Annual Meeting of the Association for Computational Linguistics*.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023. Compositional exemplars for in-context learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 39818–39833. PMLR.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. 2019. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2023. Generate rather than retrieve: Large language models are strong context generators. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*.

Yury Zemlyanskiy, Michiel de Jong, Joshua Ainslie, Panupong Pasupat, Peter Shaw, Linlu Qiu, Sumit Sanghai, and Fei Sha. 2022. Generate-and-retrieve: use your predictions to improve retrieval for semantic parsing.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. 2020. Rtfm: Generalising to new environment dynamics via reading. In *International Conference on Learning Representations*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

A Details of the datasets

Statistics on the gSCAN, GSRR and ReaSCAN dataset are reproduced in Table 7 for the reader's convenience.

	N	length ± std. (max)		N	length ± std. (max)		N	length ± std. (max)
A	19282	13.3 ± 8.9 (69)	A1	22057	17.5 ± 12.6 (92)	I	30492	8.2 ± 3.4 (24)
B	18718	14.0 ± 9.7 (72)	A2	81349	17.6 ± 13.0 (102)	II	6285	8.1 ± 3.4 (24)
C	37436	14.1 ± 9.8 (64)	B1	10002	17.4 ± 12.8 (92)	III	41576	7.8 ± 3.1 (21)
D	88642	18.0 ± 10.8 (74)	B2	6660	17.5 ± 13.3 (94)	IV	41529	8.3 ± 3.5 (24)
E	16808	13.3 ± 9.5 (54)	C1	8375	17.5 ± 13.3 (94)	TR	259088	8.1 ± 3.4 (24)
F	11460	16.5 ± 12.4 (74)	C2	8375	17.5 ± 12.9 (92)			
G	112880	33.5 ± 16.9 (104)						
H	38582	43.1 ± 19.7 (104)						
TR	367933	14.4 ± 10.1 (74)						

Table 7: Statistics on the gSCAN, ReaSCAN and GSRR datasets and training (TR) and test splits

A brief description of each split is given below:

- gSCAN A: In-distribution test split of gSCAN.
- gSCAN B: Target object is a yellow square. Yellow squares can be the target in the training split, but are not explicitly referred to as such in the instruction.
- gSCAN C: Target object is a red square. Red squares are never the target in the training split.
- gSCAN D: Target object is southwest of the agent. The agent needs to walk to the south and then to the west to reach the goal object. This sequence of actions is never seen in the training data.
- gSCAN E: "Small circles" are of size 2, which is not seen in the training data.
- gSCAN F: Action is "push" and the object size is 3, meaning that the object must be pushed twice as much as objects of size 1 and 2. Pushing an object twice as much is seen for object size 4, but not for size 3. Pulling an object twice as much as seen for size 3.
- gSCAN G: Modifier is "cautiously". This is only seen one time in the entire dataset.
- gSCAN H: Action is "pull" and modifier is "while spinning". Agent must spin on each step and then pull the object, spinning after each pull. Such a thing is seen with "push", but not "pull".
- GSRR I: In-distribution test split of GSRR.
- GSRR II: Red squares are the target object or reference object (eg, something may be *northwest* of a red square). This is not seen in the training data.
- GSRR III: A green square is a target object relative to some reference blue circle (eg, a green square may be *southeast* of a blue circle). This combination of green squares and blue circles is not seen in the training data.
- GSRR IV: Yellow-squares are the target object, even though they are never the target in the training data.
- GSRR V: Targets are north of the reference object. Targets that are northwest or northeast are seen in the training data, but not a target that is only north.
- GSRR VI: Targets are south-west of the reference object.
- ReaSCAN A1: Yellow squares are in the instruction, which is not seen in the training data.
- ReaSCAN A2: Red squares are the target object, which is not seen in the training data.
- ReaSCAN B1: "Small red square" and "big blue square" never co-occur in the training data.
- ReaSCAN B2: "Same size of" and "inside of" relation co-occurrences, which never co-occur in the training data.
- ReaSCAN C1: An additional conjugation is added to the relative clauses.
- ReaSCAN C2: An additional recursive relative clause is added, for example by swapping "and" with "that is".

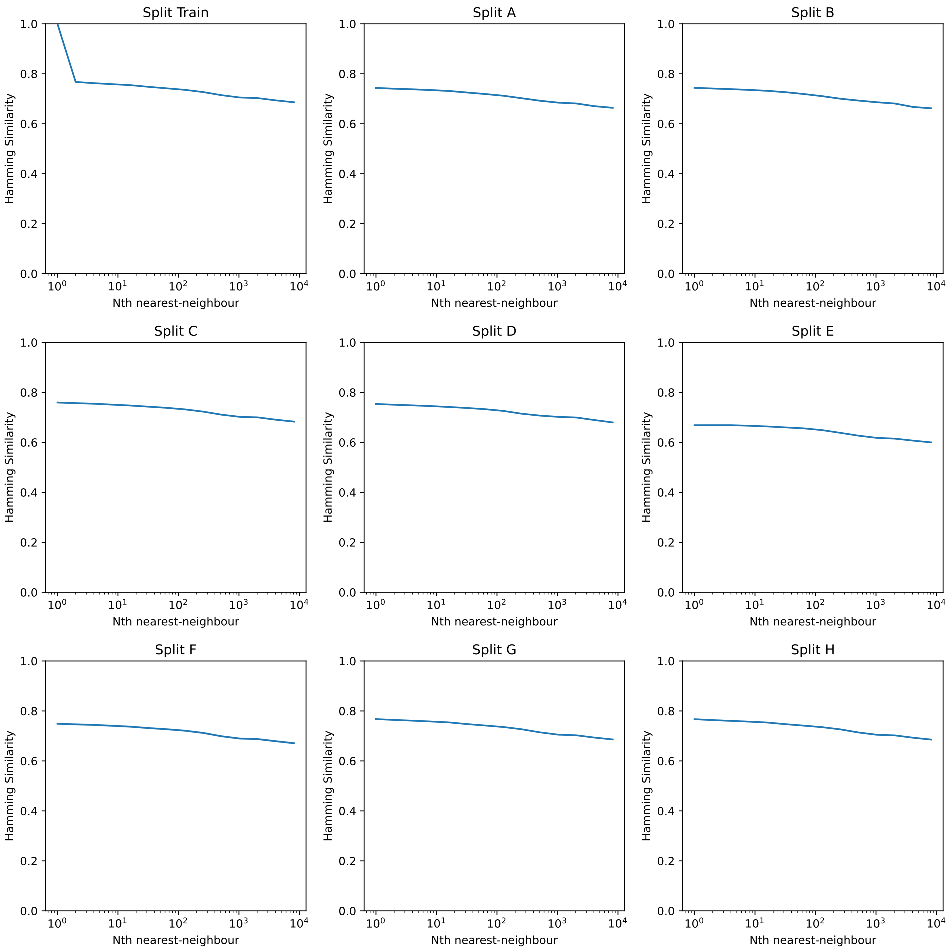


Figure 3: Average state nearest neighbour similarity (between the shown split and the training split) for each split. X-axis is log-scale. The tables show the average hamming similarity between points in a given split and their Nth nearest neighbour in the training split. TR refers to training split.

A.1 Nearest Neighbour Similarity Distribution

We visualize the average nth training-data nearest neighbour similarity distribution for each dataset split in Figure 3. We created the figure by taking 8000 random examples from each splits, then finding their 8192 nearest neighbours using a inner-product index over normalized one-hot encoded state representations.

In most cases, even the closest nearest neighbour state has quite many differences and these differences grow as we pick nearest neighbours further away from a training data point. This means that it is hard to find an example in the training set containing different instructions in the exact same environment layout. The biggest difference can be found in ReaSCAN, where even the 256th nearest neighbour can be quite a similar layout to the initial point. The reason is likely in how the dataset was generated, with a focus not so much on having many different states, but instead on having many different relational instructions in each state.

B Additional Comparisons

In this section of the appendix, we describe in more detail other related work on gSCAN and provide the results reported by those works in Table 9 for easier comparison with our experimental results.

Modular A recent work by Ruis and Lake (2022). It uses a specialized decomposition into Perception, Interaction, Navigation and Transformation Modules, each of which are trained independently with their own training outputs, then connected together at test time. The modular decomposition gives a prior on how the problem should be solved (for example by decomposition into egocentric and allocentric plans). The work also describes how data augmentation can be used to improve the model, but we show the results coming from use of the modular architecture alone. This approach can get good performance on Splits G and H. Performance on other splits is either slightly improved or comparable to the baseline in Ruis et al. (2020), which is likely due to the use of a similar underlying

	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
gSCAN														
A	0.74	0.74	0.74	0.74	0.73	0.73	0.72	0.72	0.71	0.69	0.69	0.68	0.67	0.67
B	0.75	0.74	0.74	0.74	0.73	0.73	0.72	0.71	0.70	0.70	0.69	0.68	0.67	0.67
C	0.76	0.76	0.76	0.75	0.75	0.75	0.74	0.74	0.73	0.71	0.70	0.70	0.69	0.69
D	0.75	0.75	0.75	0.75	0.74	0.74	0.74	0.73	0.72	0.71	0.70	0.70	0.70	0.69
E	0.67	0.67	0.67	0.67	0.67	0.66	0.66	0.65	0.64	0.63	0.62	0.62	0.61	0.61
F	0.75	0.75	0.75	0.74	0.74	0.73	0.73	0.73	0.72	0.70	0.69	0.69	0.68	0.68
G	0.77	0.77	0.76	0.76	0.76	0.75	0.75	0.74	0.73	0.72	0.71	0.71	0.70	0.69
H	0.77	0.77	0.76	0.76	0.76	0.75	0.75	0.74	0.73	0.72	0.71	0.71	0.70	0.69
TR	1.00	0.77	0.76	0.76	0.76	0.75	0.74	0.74	0.73	0.72	0.71	0.71	0.70	0.69
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GSRR														
I	0.64	0.63	0.62	0.61	0.61	0.60	0.59	0.59	0.58	0.57	0.56	0.55	0.53	0.52
II	0.64	0.63	0.62	0.61	0.61	0.60	0.59	0.59	0.58	0.57	0.56	0.55	0.53	0.52
III	0.63	0.63	0.62	0.61	0.61	0.60	0.59	0.59	0.58	0.57	0.56	0.55	0.53	0.52
IV	0.64	0.64	0.63	0.62	0.61	0.61	0.60	0.59	0.58	0.57	0.56	0.55	0.54	0.53
V	0.64	0.63	0.62	0.61	0.61	0.60	0.59	0.58	0.58	0.57	0.56	0.55	0.53	0.52
VI	0.64	0.63	0.62	0.61	0.61	0.60	0.59	0.59	0.58	0.57	0.56	0.55	0.53	0.52
TR	1.00	0.64	0.62	0.61	0.61	0.60	0.59	0.58	0.58	0.57	0.56	0.55	0.53	0.52
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
ReaSCAN														
A1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.97	0.97	0.97	0.97	0.97
A2	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.97	0.97	0.97	0.97	0.97
B1	0.99	0.99	0.99	0.99	0.98	0.98	0.99	0.99	0.97	0.96	0.96	0.96	0.96	0.96
B2	0.97	0.97	0.96	0.96	0.96	0.96	0.96	0.96	0.95	0.94	0.94	0.94	0.94	0.94
C1	0.98	0.97	0.97	0.97	0.96	0.97	0.97	0.97	0.96	0.95	0.95	0.95	0.94	0.94
C2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.97	0.97	0.97	0.97	0.97
TR	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.96	0.96	0.96	0.96	0.96

Table 8: Average state nearest neighbour similarity (between the shown split and the training split) for each split on gSCAN, GSRR and ReaSCAN. X-axis is log-scale. The plots show the average hamming similarity between points in a split and their Nth nearest neighbour in the training split.

architecture of RNNs and CNNs as feature encoders.

Role-Guided (Kuo et al., 2021) This approach uses linguistic priors to decompose the parsing problem and specify how sub-parsers are connected. It can achieve some level of performance on Split D and comparable performance on Split H to the Transformer.

ViLBERT is an adaptation of the ViLBERT model for gSCAN by Qiu et al. (2021) and extended on by Sikarwar et al. (2022). The state is first one-hot encoded, a few 2D convolution layers are applied to it. The state is then flattened and the channel values for each pixel are treated as vectors for each location in the state. Afterwards, there are several layers of cross-attention between the instruction tokens and the state tokens. The cross-attented representations are concatenated together and used as input to a causal Transformer decoder to decode the outputs.

GECA Also known as “Good Enough Compositional Augmentation" (Andreas (2020)), applied to gSCAN by Ruis et al. (2020). GECA is an augmentation method which recognizes *template fragments* in text, then realizes those templates with other possible substitutions. Following the example in that work, if a dataset contains “she picks the wug up in Fresno“ and “she puts the wug down in Tempe", then the augmentation method generates samples of puts down substituted into sentences containing picks up. For example the sentence “Pat picks cats up" can be augmented to “Pat puts cats down". GECA relies on being able to identify templates containing discontiguous fragments which contain at least two tokens. In the case of SCAN, GECA might identify the fragment “jump ... `JUMP ... JUMP ... JUMP`" from the concatenated instruction-action pair “jump thrice `JUMP JUMP JUMP`" and substitute it into “walk around right thrice `WALK RTURN WALK RTURN WALK RTURN`" such that it is augmented into “jump around right thrice `JUMP RTURN JUMP RTURN JUMP RTURN`". As noted by Andreas (2020), the time and space complexity of GECA can be quite large and scales with the number of recognized templates and fragments. The results reported by Ruis et al. (2020) when using GECA in Table 9 are possibly out of date, since they were generated using an RNN architecture as opposed to a Transformer, where better performance on Splits B, C, E and F has been observed. Also, GECA was only applied to the instructions and state and not to the target commands. Possibly the reason for this is that the computational and memory complexity of GECA makes it

	seq2seq	GECA	FiLM	RelNet	LCGN	ViLBERT
	(Ruis et al., 2020)	(Ruis et al., 2020)	(Qiu et al., 2021)	(Qiu et al., 2021)	(Gao et al., 2020)	(Qiu et al., 2021)
A	97.15 ± .46	87.6 ± 1.19	98.83 ± .32	97.38 ± .33	98.6 ± .9	99.95 ± .02
B	30.05 ± 26.76	34.92 ± 39.30	94.04 ± 7.41	49.44 ± 8.19	99.08 ± .69	99.90 ± .06
C	29.79 ± 17.70	78.77 ± 6.63	60.12 ± 8.81	19.92 ± 9.84	80.31 ± 24.51	99.25 ± .91
D	0.00 ± .00	0.00 ± .00	0.00 ± .00	0.00 ± .00	0.16 ± .12	0.00 ± .00
E	37.25 ± 2.85	33.19 ± 3.69	31.64 ± 1.04	42.17 ± 6.22	87.32 ± 27.38	99.02 ± 1.16
F	94.16 ± 1.25	85.99 ± .85	86.45 ± 6.67	96.59 ± .94	99.33 ± .46	99.98 ± .01
H	19.04 ± 4.08	11.83 ± .31	11.71 ± 2.34	18.26 ± 1.24	33.6 ± 20.81	22.16 ± .01
I	86.48 ± .64	88.5 ± .82	85.17 ± 3.81	-	94.66 ± .24	-
II	40.10 ± .83	50.68 ± .32	38.59 ± .74	-	64.41 ± 4.52	-
III	86.08 ± .73	88.81 ± 1.42	85.66 ± 4.35	-	94.89 ± .20	-
IV	5.47 ± .09	10.78 ± 3.47	4.85 ± .86	-	49.58 ± 3.47	-
V	81.41 ± 1.03	76.20 ± 2.64	79.86 ± 3.16	-	59.29 ± 5.63	-
VI	81.84 ± 1.38	75.05 ± 3.63	80.93 ± 2.76	-	49.50 ± 6.49	-
A1	50.36 ± 4.03	-	-	99.25 ± .77	-	-
A2	14.64 ± .55	-	-	42.05 ± 4.55	-	-
B1	52.17 ± 1.63	-	-	69.74 ± .30	-	-
B2	39.41 ± 1.53	-	-	52.80 ± 2.75	-	-
C1	49.68 ± 2.73	-	-	57.01 ± 7.99	-	-
C2	25.74 ± 1.36	-	-	22.07 ± 2.66	-	-

	GroCoT	Planning	RD Random/RL	Modular	CMA-ES	Role-Guided
	(Sikarwar et al., 2022)	2020	(Setzler et al., 2022)	(Ruis and Lake, 2022)	(Hein and Diepold, 2022)	(Kuo et al., 2021)
A	99.9	94.19 ± .71	98.39 ± .17	96.34 ± .28	99.7 ± .1	96.73 ± .58
B	99.9	87.31 ± 4.38	62.19 ± 24.08	59.66 ± 23.76	73.5 ± 25.4	94.91 ± 1.30
C	99.9	81.07 ± 10.12	56.52 ± 29.70	32.09 ± 9.79	99.4 ± .4	67.72 ± 10.83
D	0.0	-	43.60 ± 6.05	0.00 ± .00	2.2 ± 1.5	11.52 ± 8.18
E	99.8	52.8 ± 9.96	53.89 ± 5.39	49.34 ± 11.60	97.4 ± 2.0	76.83 ± 2.32
F	99.9	-	95.74 ± .75	94.16 ± 1.25	99.1 ± .6	98.67 ± .05
H	22.9	-	21.95 ± .03	76.84 ± 26.94	98.4 ± 1.1	20.98 ± 1.98
I	99.8	-	-	-	-	-
II	98.6	-	-	-	-	-
III	99.9	-	-	-	-	-
IV	99.7	-	-	-	-	-
V	99.5	-	-	-	-	-
VI	96.5	-	-	-	-	-
A1	99.6	-	-	-	-	-
A2	93.1	-	-	-	-	-
B1	93.9	-	-	-	-	-
B2	86.0	-	-	-	-	-
C1	76.3	-	-	-	-	-
C2	27.3	-	-	-	-	-

Table 9: Additional related work comparisons on gSCAN, GSRR and ReaSCAN Splits G and I are not included.

difficult to apply the joint space of the state, instruction and target commands as in gSCAN.

CMA-ES uses sparse hard attention with CMA-ES as the optimization algorithm as opposed to a gradient-based optimizer. The model architecture consists only of a multi-layer perceptron, predicting the next token with attention over the generated output sequence. The method requires some supervision on what the goal object is, in contrast with other approaches. Its strengths are that convergence can happen very quickly and optimization can be run on lighter hardware. The method also gets very good performance on Split H, however, as of the time of writing, the authors have not yet published their code and did not provide any analysis in their paper as to why the measured Split H performance was so good, so it is not possible to make a detailed comparison with our work.

	ViLBERT	Modular	Role-guided	Transformer (ours)	ICL Transformer
	(Qiu et al., 2021)	(Ruis and Lake, 2022)	(Kuo et al., 2021)	Ours	Ours
Learning Rate	0.0015	0.001	0.001	0.0001	0.0001
Embedding Dim	128	128	128	512	512
Dropout	0.1	-	-	0.1	0.1
Batch Size	128	200	200	128	128
Steps	114.96K	73K	150K	300K	300K
#params	3M			88.3M	88.3M

Table 10: Hyperparameters used in our experiments and the related work

C Experimental Details

We ran experiments to determine the performance of our approach. The Transformer blocks use an embedding size (d_{model}) of 512 units and fully-connected layer size (d_{FF}) of 2048 units is used. We use 12 layers for each of the encoder and decoder of the encoder-decoder transformer. The learning rate is 10^{-5}, we have an effective batch size of 128, and training iteration count of 300,000. During training, dropout is not used and weight decay is set to 10^{-3} with the AdamW optimizer. Beta values are left at their defaults, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Learning rate warmup is used up to step 30,000 to a peak learning rate of 10^{-5}, then decayed on a log-linear schedule from steps 30,000 to 300,000 to 10^{-6}. Gradient norms are clipped at 0.2 to improve training stability. We use 16-bit precision during training and make use of gradient accumulation in order to simulate large batch sizes where memory is limited.

D Implementation of GandR for grounded language datasets

We make small adaptations to GandR (Zemlyanskiy et al., 2022) to adapt it to the grounded setting. The baseline transformer model makes an initial prediction for the query input, then the query input and prediction are vector-encoded (the instruction using the `sentence-transformers` package and the actions using TF-IDF) and used to find other similar query-output pairs using the index, which become the support inputs and outputs used for ICL. States are encoded using a PCA projection of their sparse representations. Compared to the original, we keep the α trade-off between input and target components fixed as opposed to varying it. There is also nothing to ensure that a diversity of different instructions is sampled - only the near neighbours are sampled, even if they all correspond to a single instruction.

E Implementation of CovR for grounded language datasets

We implement the main idea behind Set-BSR (Gupta et al., 2023) for the grounded setting. States are vector-encoded and projected using PCA into 320 dimensions. Instructions are encoded using the `sentence-transformers` package. Both are concatenated with each other to make a vector representation of an example. The instruction component of the vector is weighted with $\alpha = 0.125$. The training-set vectors are placed into an inner-product index. For performance reasons, we use a Voronoi index with 512 cells and 10 cell probes per search. For each vector in a split, we search the index for the 128 nearest neighbours, sort the neighbours in descending order according to the number of matching two-grams, one-grams and the cosine similarity to the query state. Then we pick the top $k = 16$ examples as the support set.

F Properties of Generated Demonstrations, other splits

Properties of Generated Demonstrations for the other splits are shown in tables below.

G Heuristic Function

The **Heuristic** function generates relevant instructions by the use of a templating mechanism, which replaces verbs and adverbs in the sentence with other verbs and adverbs, such that the whole combination is still in distribution, but not the same as the query instruction. The rules of the system are:

- Replace “pull" with “push" and “walk to"
- Replace “walk to" with “push" and “pull" (but not if “while spinning" is the adverb)
- Replace “push" with “walk to" and “pull" (but not if “while spinning" is the adverb)
- Replace “while zigzagging" with “hesitantly", nothing and “while spinning" (but not if “push" is the verb)
- Replace “hesitantly" with “while zigzagging", nothing and “while spinning" (but not if “push" is the verb)
- Replace “while spinning" with “hesitantly", “while zigzagging" and nothing

Examples of what the oracle function generates for a given query instruction and environment can be found in Figure 10. Actions are generated by using the same procedure provided in Ruis et al. (2020). The instruction generated by the oracle is given to the demonstration generation procedure and a demonstration is generated by that. A demonstration can also be generated by providing the oracle-generated instruction and current state representation as input to a Transformer model trained on the provided training set.

H Permuter Blocks

The permuter block shuffles the indices mapping words to symbols in the dictionary given in Table 11. Table 12 gives an example of how the permuted sequences might look to the encoders. Essentially the individual symbols no longer hold any special meaning without reference to the demonstrations, only conditional autoregressive probabilities up to a permutation hold meaning.

Split A						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.32	0.83	0.15	1.00	1.00	0.07
(2) Agent Pos.	1.00	0.07	1.00	1.00	0.03	1.00
(3) Tgt. Pos.	0.37	0.08	0.27	1.00	0.03	0.07
(4) Same Diff.	0.37	0.31	0.27	1.00	0.02	0.07
(5) Tgt. Obj.	0.37	0.26	0.22	1.00	0.25	0.07
(6) Verb & (5)	1.00	0.93	0.91	1.00	0.50	0.07
(7) Advb & (5)	0.75	0.93	0.77	1.00	0.38	0.07
(8) (6) & (7)	0.75	0.93	0.73	1.00	0.23	0.07
(9) (4) & (8)	0.75	0.57	0.65	1.00	0.00	0.07

Split B						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.26	0.00	0.00	1.00	0.00	0.00
(2) Agent Pos.	1.00	0.13	1.00	1.00	0.00	1.00
(3) Tgt. Pos.	0.32	0.15	0.29	1.00	0.00	0.00
(4) Same Diff.	0.32	0.44	0.29	1.00	0.00	0.00
(5) Tgt. Obj.	0.32	0.03	0.18	1.00	0.00	0.00
(6) Verb & (5)	1.00	0.30	0.85	1.00	0.00	0.00
(7) Advb & (5)	0.66	0.30	0.71	1.00	0.00	0.00
(8) (6) & (7)	0.66	0.30	0.69	1.00	0.00	0.00
(9) (4) & (8)	0.66	0.24	0.63	1.00	0.00	0.00

Split C						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.16	0.47	0.15	1.00	1.00	0.15
(2) Agent Pos.	1.00	0.12	1.00	1.00	0.03	1.00
(3) Tgt. Pos.	0.19	0.13	0.18	1.00	0.03	0.15
(4) Same Diff.	0.19	0.44	0.18	1.00	0.02	0.15
(5) Tgt. Obj.	0.19	0.00	0.00	1.00	0.00	0.15
(6) Verb & (5)	0.79	0.00	0.00	1.00	0.00	0.15
(7) Advb & (5)	0.41	0.00	0.00	1.00	0.00	0.15
(8) (6) & (7)	0.40	0.00	0.00	1.00	0.00	0.15
(9) (4) & (8)	0.40	0.00	0.00	1.00	0.00	0.15

Split D						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.19	0.83	0.18	1.00	1.00	0.16
(2) Agent Pos.	1.00	0.03	1.00	1.00	0.02	1.00
(3) Tgt. Pos.	0.33	0.03	0.00	1.00	0.02	0.16
(4) Same Diff.	0.33	0.00	0.00	1.00	0.00	0.16
(5) Tgt. Obj.	0.33	0.20	0.05	1.00	0.10	0.16
(6) Verb & (5)	0.99	0.89	0.42	1.00	0.25	0.16
(7) Advb & (5)	0.89	0.88	0.25	1.00	0.17	0.16
(8) (6) & (7)	0.89	0.88	0.20	1.00	0.06	0.16
(9) (4) & (8)	0.89	0.00	0.00	1.00	0.00	0.16

Split E						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.22	0.89	0.07	1.00	0.00	0.00
(2) Agent Pos.	1.00	0.11	1.00	1.00	0.00	1.00
(3) Tgt. Pos.	0.27	0.12	0.22	1.00	0.00	0.00
(4) Same Diff.	0.27	0.35	0.22	1.00	0.00	0.00
(5) Tgt. Obj.	0.27	0.03	0.14	1.00	0.00	0.00
(6) Verb & (5)	0.96	0.20	0.81	1.00	0.00	0.00
(7) Advb & (5)	0.50	0.20	0.63	1.00	0.00	0.00
(8) (6) & (7)	0.50	0.20	0.60	1.00	0.00	0.00
(9) (4) & (8)	0.50	0.14	0.50	1.00	0.00	0.00

Split F						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.26	0.81	0.23	1.00	1.00	0.15
(2) Agent Pos.	1.00	0.12	1.00	1.00	0.03	1.00
(3) Tgt. Pos.	0.33	0.15	0.26	1.00	0.03	0.15
(4) Same Diff.	0.33	0.37	0.26	1.00	0.02	0.15
(5) Tgt. Obj.	0.33	0.00	0.10	1.00	0.07	0.15
(6) Verb & (5)	0.96	0.00	0.00	1.00	0.00	0.15
(7) Advb & (5)	0.60	0.00	0.62	1.00	0.29	0.15
(8) (6) & (7)	0.58	0.00	0.00	1.00	0.00	0.15
(9) (4) & (8)	0.58	0.00	0.00	1.00	0.00	0.15

Split G						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.39	0.91	0.31	1.00	1.00	0.20
(2) Agent Pos.	1.00	0.14	1.00	1.00	0.03	1.00
(3) Tgt. Pos.	0.50	0.16	0.37	1.00	0.03	0.20
(4) Same Diff.	0.50	0.35	0.37	1.00	0.02	0.20
(5) Tgt. Obj.	0.50	0.22	0.24	1.00	0.20	0.20
(6) Verb & (5)	1.00	0.91	0.93	1.00	0.51	0.20
(7) Advb & (5)	0.00	0.01	0.00	1.00	0.00	0.20
(8) (6) & (7)	0.00	0.01	0.00	1.00	0.00	0.20
(9) (4) & (8)	0.00	0.00	0.00	1.00	0.00	0.20

Split H						
	DemoG	GandR	CovR	Expert	OS	RD
(1) Desc. Obj.	0.33	0.68	0.33	1.00	1.00	0.16
(2) Agent Pos.	1.00	0.08	1.00	1.00	0.03	1.00
(3) Tgt. Pos.	0.44	0.08	0.39	1.00	0.03	0.16
(4) Same Diff.	0.44	0.09	0.39	1.00	0.02	0.16
(5) Tgt. Obj.	0.44	0.14	0.27	1.00	0.19	0.16
(6) Verb & (5)	1.00	0.15	0.88	1.00	0.43	0.16
(7) Advb & (5)	0.88	0.51	0.78	1.00	0.33	0.16
(8) (6) & (7)	0.88	0.00	0.70	1.00	0.19	0.16
(9) (4) & (8)	0.88	0.00	0.62	1.00	0.00	0.16

I Natural-ish Language gSCAN Dataset

The dataset is generated by extracting all of the input sentences from gSCAN and its derivatives, then using the commercial `gpt3.5-turbo` model from OpenAI[2] to generate additional paraphrases of the input sentence. The paraphrases are generated by creating four dataset specific prompts, each with an 10 examples of how one instruction in the dataset may be paraphrased, then requesting 25 additional paraphrases for a different instruction in the same dataset to be completed by the language model. The prompts are given in Appendix J. The prompts modes are described as follows:

Simple Paraphrases of “Push a red square"

Adverb Paraphrases of “Push a red square cautiously"

Relational Paraphrases of “Push a red circle that is south east of a blue circle"

ReaSCAN Paraphrases of “Pull the yellow square that is inside of a big red box and in the same row as a small red circle and in the same column as a small cylinder while spinning"

The 10 paraphrase examples were written by ourselves - the idea is that they show how adverbs and actions can be replaced by synonyms, and also show examples of the same instruction in a different sentence ordering. For

[2] As of 5 May 2023

Word	Symbol	Action	Symbol
‘a’	0	PULL	0
‘big’	1	PUSH	1
‘blue’	2	STAY	2
‘cautiously’	3	LTURN	3
‘circle’	4	RTURN	4
‘cylinder‘	5	WALK	5
‘green’	6		
‘hesitantly’	7		
‘pull’	8		
‘push	9		
‘red’	10		
‘small’	11		
‘square’	12		
‘to’	13		
‘walk’	14		
‘while spinning’	15		
‘while zigzagging‘	16		

Table 11: Default mapping of words and actions to symbols

example, “push a red square" can be paraphrased as “shove the red square" or “Walk to a red square and push it". The paraphrases can also include additional verbs adverbs which are distractors, for example “grasp a red square and move it along".

We generate paraphrases of instructions in gSCAN, GSRR and ReaSCAN. The default generation mode creates paraphrases for each unique instruction individually. However for GSRR and ReaSCAN, the number of unique instructions is very large, which would mean that generation would come at both a high time and monetary cost. The reason for this is the combinatorial explosion of factors; in GSRR the instructions are given as target objects with positions relative to other objects. To address this problem, we also have a "template" generation mode, which replaces the object descriptions (size, color, shape) with a template placeholder, generates paraphrases for the templates, then generates “realised" paraphrases from those templates for each of the corresponding object descriptions. This reduces the number of requests to the model from hundreds of thousands to thousands.

J Prompts used for GPT3.5

J.1 gSCAN Simple Prompt

Here are 10 similar statements to “push a red square"

1. Push the red square
2. Move a red square
3. Shove the red square
4. Go to the red square and shove it
5. Go to the red square and push it
6. Walk to the red square and push it
7. Find a red square and push it
8. Locate a red square and push it
9. Get to the red square and move it along
10. Walk up to the red square and then really push it

Can you generate 25 similar statements for “{{QUERY}}” in English?

K Examples of Generated Paraphrases

Examples: 367933 Unique Instructions: 430 Templates: 828 Sample Responses:

Original actions	Permutation	Encoded actions	Permuted encoding
WALK(5) RTURN WALK(5)	PULL(0) → 0, PUSH(1) → 5, STAY(2) → 2, LTURN(3) → 1, RTURN(4) → 3, WALK(5) → 4,	5(5) 4 5(5)	4(5) 3 4(5)
RTURN WALK(3)	PULL(0) → 0, PUSH(1) → 2, STAY(2) → 3, LTURN(3) → 5, RTURN(4) → 4, WALK(5) → 1,	4 5(3)	4 1(3)
LTURN(4) WALK LTURN(4) WALK LTURN(5) WALK LTURN(4) WALK LTURN(4) WALK LTURN(4) WALK LTURN(4) WALK	PULL(0) → 4, PUSH(1) → 5, STAY(2) → 0, LTURN(3) → 2, RTURN(4) → 3, WALK(5) → 1,	3(4) 5 3(4) 5 3(5) 5 3(4) 5 3(4) 5 3(4) 5 3(4) 5	2(4) 1 2(4) 1 2(5) 1 2(4) 1 2(4) 1 2(4) 1 2(4) 1
LTURN WALK STAY WALK STAY WALK STAY WALK STAY	PULL(0) → 3, PUSH(1) → 0, STAY(2) → 2, LTURN(3) → 5, RTURN(4) → 4, WALK(5) → 1,	3 5 2 5 2 5 2 5 2	5 1 2 1 2 1 2 1 2
LTURN WALK STAY WALK STAY	PULL(0) → 0, PUSH(1) → 3, STAY(2) → 4, LTURN(3) → 5, RTURN(4) → 2, WALK(5) → 1,	3 5 2 5 2	5 1 4 1 4
LTURN(4) WALK LTURN(4) WALK LTURN(4) WALK LTURN(4) RTURN WALK LTURN(4) WALK LTURN(4) WALK LTURN(4) WALK LTURN(4) WALK	PULL(0) → 0, PUSH(1) → 4, STAY(2) → 5, LTURN(3) → 1, RTURN(4) → 3, WALK(5) → 2,	3(4) 5 3(4) 5 3(4) 5 3(4) 4 5 3(4) 5 3(4) 5 3(4) 5 3(4) 5	1(4) 2 1(4) 2 1(4) 2 1(4) 3 2 1(4) 2 1(4) 2 1(4) 2 1(4) 2
LTURN WALK(2) PUSH	PULL(0) → 1, PUSH(1) → 0, STAY(2) → 5, LTURN(3) → 3, RTURN(4) → 4, WALK(5) → 2,	3 5(2) 1	3 2(2) 0

Table 12: Actions and possible mapping permutations generated by the permuter block.

	gSCAN	RS	ReaSCAN
Uniq. Instrs.	430	31799	4381
Uniq. Tmpls.	-	21	658
Gen. Instrs.	12778	731377	99698
Gen. Tmpls.	-	483	14683
Prompt	Simple	Relational	ReaSCAN

Table 13: Generation properties and configuration for each of the datasets

Original walk to a yellow small cylinder hesitantly

1. Walk hesitantly to a small yellow cylinder
2. Approach a small yellow cylinder with hesitation
3. Take cautious steps towards a small yellow cylinder
4. Walk slowly towards a small yellow cylinder
5. Move towards a small yellow cylinder with caution
6. Walk towards a small yellow cylinder carefully
7. Take hesitant steps towards a small yellow cylinder
8. Approach a small yellow cylinder slowly and hesitantly
9. Walk towards a small yellow cylinder with apprehension
10. Move towards a small yellow cylinder hesitantly
11. Walk to a small yellow cylinder with reluctance
12. Take tentative steps towards a small yellow cylinder
13. Walk towards a small yellow cylinder with uncertainty
14. Approach a small yellow cylinder with trepidation
15. Walk towards a small yellow cylinder with reservation

16. Take hesitant strides towards a small yellow cylinder
17. Walk to a small yellow cylinder with caution
18. Move towards a small yellow cylinder with unease
19. Walk towards a small yellow cylinder with doubt
20. Approach a small yellow cylinder with timidity
21. Walk towards a small yellow cylinder with hesitance
22. Take slow steps towards a small yellow cylinder
23. Walk towards a small yellow cylinder with wariness
24. Move towards a small yellow cylinder with hesitancy
25. Walk towards a small yellow cylinder with reluctance and caution.

Original pull a circle hesitantly

1. Hesitantly pull a circle
2. Pull a circle with hesitation
3. Gently pull a circle
4. Pull a circle slowly and cautiously
5. Approach the circle and pull it hesitantly
6. Pull the circle with care
7. Pull the circle slowly and carefully
8. Take your time and pull the circle hesitantly
9. Pull the circle with a bit of hesitation
10. Approach the circle and pull it gently
11. Pull the circle with caution
12. Pull the circle carefully and hesitantly
13. Slowly pull the circle with hesitation
14. Pull the circle with a sense of caution
15. Pull the circle with a bit of reluctance
16. Pull the circle slowly and with care
17. Hesitate before pulling the circle
18. Pull the circle with a gentle touch
19. Pull the circle with a bit of apprehension
20. Pull the circle with a sense of uncertainty
21. Pull the circle with a bit of nervousness
22. Pull the circle with a bit of trepidation
23. Pull the circle with a sense of hesitation
24. Pull the circle with a bit of doubt
25. Pull the circle with a bit of reservation

Original pull a circle while zigzagging

1. Zigzag while pulling a circle
2. Pull a circle in a zigzag pattern
3. Carefully pull a circle while zigzagging
4. Zigzag and pull a circle simultaneously
5. Pull a circle while moving in a zigzag motion
6. With caution, pull a circle while zigzagging
7. Zigzag your way to the circle and pull it
8. Pull a circle while making zigzag movements
9. Zigzag and pull the circle with care
10. Pull a circle while navigating in a zigzag direction
11. Move in a zigzag pattern while pulling a circle
12. Pull a circle while making a zigzag path
13. Zigzag towards the circle and pull it
14. Pull a circle while making zigzag turns
15. Carefully zigzag and pull the circle
16. Zigzag and carefully pull the circle
17. Pull a circle while making sharp zigzag movements
18. Zigzag and pull the circle with caution
19. Pull a circle while making quick zigzag motions
20. Zigzag and pull the circle slowly
21. Pull a circle while zigzagging in a controlled manner
22. Zigzag and pull the circle with precision
23. Pull a circle while making small zigzag movements
24. Zigzag and pull the circle with care and attention
25. Pull a circle while zigzagging smoothly.

L Properties of Natural-ish Language gSCAN Dataset

	parses	words	zipf a	rmse
gSCAN	18	18	1.99	0.11
NL-gSCAN	1550	859	1.29	0.01
GSRR	234	20	1.90	0.10
NL-GSRR	9785	126	1.40	0.03
ReaSCAN	1400	35	1.26	0.04
NL-ReaSCAN	42759	631	1.22	0.01

Figure 4: Linguistic properties of each dataset and its corresponding paraphrased (denoted NL-) dataset
.

	Size	Color	Object
gSCAN	100%	99.98%	98.63%
SR	100%	100%	100%
ReaSCAN	100%	99.99%	99.93%

Figure 5: Percentage of examples in each training set whether the object mentioned in the synthetic dataset was also found in exactly the same way the corresponding paraphrased example.

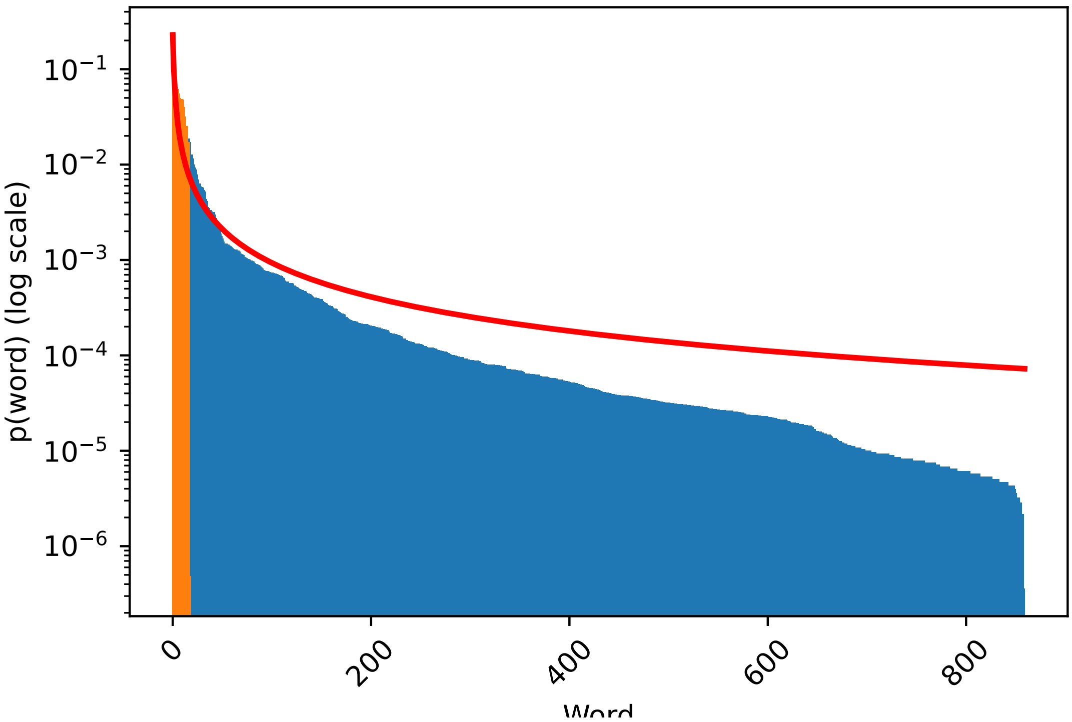


Figure 6: Word frequency distribution of NL-gSCAN and gSCAN, each compared to the best fitting Zipf distribution probability density function. gSCAN words are in orange and NL-gSCAN words are in blue (comprising of the larger vocabulary).

L.1 Linguistic Properties

In this section we examine the linguistic properties of the dataset. The main research question is whether the instructions as paraphrased by GPT3.5 look more like natural language. Clearly, the paraphrased data has greater vocabulary complexity. But merely substituting words for synonyms would not make synthetic data appear any more natural, nor does it pose any real challenges to a learning algorithm that would need to act on the instructions. We examine two other indicia, unique parses and fit to a Zipf distribution of word frequency.

Parses We compute the number of unique parses among all the instructions in each training set. A *parse* is an assignment of word-role labels, indicating the linguistic role of the token in the instruction. For example, a token may be an adjective, an adverb or some sort of connector. The parses are computed over every instruction in the training data using the spaCy package. As shown in Table 4, the number of unique parses in the paraphrased datasets are an order of magnitude larger than the number of unique parses in the synthetic datasets. This reflects the diversity of instruction structures that exist in the paraphrased datasets.

Zipfian Distribution Fit Natural language is hypothesized to fit a Zipfian power-law distribution, where the probability of drawing a word from a corpus is inversely proportional to its frequency $p(w) \propto \frac{1}{f_w^a}$, where a is a parameter of the distribution which varies for different corpii. We estimate a using maximum likelihood estimation using the method in (Clauset et al., 2009) and compute the root-mean-squared error (RMSE) between the estimated probability of a word according to the estimated Zipf distribution and the empirical probability that word measured by counting word frequencies. A corpus that resembles natural language more closely will have a low RMSE to its correpsonding Zipf distribution. We find that the paraphrased datasets better fit their Zipf distribution. We also visualize in both Figure 6 the ordered frequency distribution of the paraphrased gSCAN dataset and its corresponding Zip probability density function.

L.2 Compositional Properties

We also examine whether the datasets maintained their compositional properties. Recall that the datasets are stratified into different splits to test different compositional generalization cases. We want to test whether these cases still hold. Clearly, in the output space, the compositional stratification still holds because we do not change the output actions. In the input space, we can only measure whether the same object is mentioned in each synthetic instruction and its corresponding paraphrased instruction, because the verbs and adverbs may be changed to a synonym or a sequence of words having a similar meaning.

In all three datasets, the retainment of target objects is very high, never going under 98%. We can be confident that the correct target object is mentioned in the same way in the paraphrased examples.

M Evaluation of baselines on Natural-ish gSCAN, GSRR and ReaSCAN

We evaluate current published state-of-the-art models with openly available code on the new datasets using our own re-implementation. We calculate the exact-match performance using seeds 0-9 using the same hyperparameters for each model, the details of which are specified in Appendix B. The models are briefly described below:

ViLBERT with Cross-Attention The ViLBERT model proposed in (Qiu et al., 2021), with only cross-attention between visual and text input streams, then decoding the target action sequence autoregressively. As in (Sikarwar et al., 2022), the multi-level CNN on the grid world is replaced by adding learnable position encodings.

Encoder-Decoder Transformer A standard encoder-decoder Transformer, where the transformer input sequence is the position-encoded and embedded visual stream concatenated with the instruction, and the target output sequence are the actions, decoded autoregressively.

	Transformer	ViLBERT	ViLBERT(PP)
		gSCAN	
A	1.0 ± .00	1.0 ± .00	1.0 ± .00
B	0.86 ± .28	0.94 ± .11	0.93 ± .09
C	0.89 ± .16	0.89 ± .13	0.82 ± .26
D	0.01 ± .02	0.0 ± .01	0.0 ± .00
E	0.99 ± .02	0.93 ± .12	0.71 ± .24
F	1.0 ± .00	1.0 ± .00	1.0 ± .00
G	0.0 ± .00	0.0 ± .00	0.0 ± .00
H	0.19 ± .06	0.23 ± .01	0.17 ± .06
		GSRR	
I	1.0 ± .00	1.0 ± .00	1.0 ± .00
II	0.95 ± .04	0.93 ± .04	0.96 ± .02
III	0.99 ± .01	0.96 ± .03	1.0 ± .00
IV	1.0 ± .00	1.0 ± .00	1.0 ± .00
V	0.46 ± .26	0.72 ± .1	0.9 ± .04
VI	0.17 ± .18	0.61 ± .23	0.89 ± .06
		ReaSCAN	
IID	0.99 ± .00	0.98 ± .02	0.97 ± .01
A1	0.94 ± .02	0.95 ± .04	0.95 ± .01
A2	0.61 ± .05	0.52 ± .13	0.46 ± .07
B1	0.75 ± .02	0.79 ± .05	0.75 ± .03
B2	0.54 ± .02	0.6 ± .09	0.53 ± .05
C1	0.37 ± .02	0.32 ± .02	0.64 ± .03
C2	0.27 ± .05	0.22 ± .05	0.22 ± .03

Figure 7: The evaluation results for gSCAN, GSRR and ReaSCAN at 300,000 iterations, where performance for splits B-H is measured at the point where the model performed best on split A during training. ViLBERT is the model in (Qiu et al., 2021) and Transformer is an Encoder-Decoder Transformer. Tformer(PP) the same Transformer architecture evaluated on the paraphrased dataset.

N Image-Based gSCAN

We also created an Image-Based gSCAN dataset where the state inputs are images instead of integer-encoded tilemaps. The model is adjusted to be similar to the Vision Transformer (Dosovitskiy et al., 2021) with a patch size of 12. DemoGen is implemented in the same way, by first training a model on the base dataset that uses image-based state representations, then by generating demonstrations from those images, then by using those demonstrations and the patch-encoded images as examples in a second meta-learning transformer module. The results are reported in Table 8. We observed a similar boost on Split H for the NL + Img dataset as well. However, we note that the model for NL + Img appeared to be underfitting, so it is possible that with a larger model that the results could have been even better.

	Transformer		DemoGen	
	NL	+Img	NL	+Img
A	1.0 ± .00	1.0 ± .00	0.99 ± .00	0.84 ± .01
B	0.99 ± .00	0.93 ± .08	0.96 ± .00	0.53 ± .01
C	0.99 ± .03	0.89 ± .16	0.97 ± .00	0.54 ± .01
D	0.08 ± .16	0.0 ± .00	0.01 ± .01	0.11 ± .02
E	0.98 ± .03	0.83 ± .22	0.98 ± .00	0.67 ± .00
F	1.0 ± .00	1.0 ± .00	0.98 ± .00	0.88 ± .01
G	0.0 ± .00	0.0 ± .00	0.0 ± .00	0.0 ± .00
H	0.19 ± .03	0.06 ± .05	0.59 ± .06	0.48 ± .02

Figure 8: Evaluation on natural language and image data. NL refers to natural language instructions, NL + Img refers to natural language instructions and patch-encoded images

O Evaluating on LLMs

We also fine-tuned LLaMA3-Instruct using LoRA on training data from gSCAN, GSRR and ReaSCAN. Because LLaMA3 is a language model and gSCAN uses symbolic inputs for the state, we "encode" the state as text, by giving it as a description. We found that fine-tuning was necessary - few-shot evaluation using the both the generated and retrieved examples on ChatGPT was very poor, with the model often hallucinating actions. We compare both the ICL and non-ICL problem formulations after fine-tuning the model on both ICL datasets and non-ICL datasets.

Examples of encoded inputs are provided in Table 9. The results of the evaluation (exact match performance) are provided in Table 14. We do not use the symbol-index permutation as a means to support meta-learning, but instead rely on the previously demonstrated capability of large language models to do in-context few-shot learning. While the results are not as good as using the meta-ICL transformer with symbol-index permutation in Table 2, the ICL cases clearly outperform the baseline where we only finetuned LoRA on the original dataset reformatted as text. In these cases, DemoGen is very competitive, again getting superior performance on gSCAN split H, NL-gSCAN split H and performing competitively on both GSRR and ReaSCAN.

P Performance and similarity of generated examples

In Table 15 we examine the relationship between the relevance of the supports instructions to the query instruction and performance and also the diversity within the support instructions and performance. The *relevance* of a support instruction to the query is measured as the inner product of the normalized embeddings of instructions as produced by the `sentence-transformers` package using the `all-mpnet-base-v2` model. The diversity of support is measured as the mean value of the upper triangle of the all-pairs normalized euclidean distances (normalized to be between 0 and 1), as given by:

$$\sum \frac{\text{triu}(||\hat{E} - \hat{E}^T)||_2^2}{N(N-1)/2} \tag{1}$$

where E is the matrix of `sentence-transformer` embedded support inputs. This value will be 1 where all supports are completely different from each other and 0 where they are completely the same.

gSCAN splits c and h are shown as these are the splits where the performance was not either very close to 100% or very close to 0. We found that there was a weak correlation between relevance and exact match performance. This is reflected in the histogram of support relevance bins and their corresponding exact match performance value, where it can be seen that mean exact match performance roughly increases alongside the relevance. With intra-support diversity, the story is a bit different. The overall correlation is negative, however there is a curve where increased diversity between the supports from 0.6 to 0.73 comes with marginally improved performance, but then that performance drops off once diversity starts to increase from 0.7 to 0.89 (performance dropping from 88% exact match performance to 60% exact match performance).

Q Examples of generated demonstrations

We provide one-example-per-method of each support generation method on Split H in Figure 10. Examples in green are valid in the environment, relevant to the target object and correctly executed. Examples in yellow are considered

	Original				Paraphrased			
gSCAN	Baseline	CR	GR	DG	Baseline	CR	GR	DG
A	0.00	0.32	0.75	**0.73**	0.00	0.08	0.54	**0.62**
B	0.01	0.51	0.88	**0.97**	0.01	0.17	0.45	**0.48**
C	0.02	0.34	0.87	**0.97**	0.02	0.10	0.45	**0.44**
D	0.00	0.00	0.0	**0.17**	0.00	0.00	0.0	**0.27**
E	0.03	0.55	0.81	**0.97**	0.02	0.20	0.56	**0.71**
F	0.01	0.30	0.69	**0.84**	0.02	0.12	0.51	**0.72**
G	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0
H	0.00	0.06	0.15	**0.60**	0.00	0.04	0.13	**0.30**
GSRR								
I	0.01	0.35	0.52	**0.95**	0.01	0.20	0.53	**0.99**
II	0.02	0.15	0.54	**0.79**	0.01	0.11	0.50	**0.76**
III	0.02	0.10	0.50	**0.86**	0.01	0.10	0.46	**0.83**
IV	0.01	0.32	0.54	**0.89**	0.01	0.22	0.52	**0.99**
V	0.01	0.45	0.48	**0.65**	0.00	0.23	0.50	**0.79**
VI	0.02	0.45	0.48	**0.71**	0.01	0.22	0.49	**0.76**
ReaSCAN								
A1	0.00	0.00	0.37	**0.41**	0.00	0.00	0.35	**0.74**
A2	0.00	0.00	0.34	**0.49**	0.00	0.00	0.31	**0.74**
B1	0.00	0.00	0.35	**0.49**	0.00	0.00	0.35	**0.66**
B2	0.00	0.00	0.22	0.22	0.00	0.00	0.24	**0.40**
C1	0.00	0.00	0.10	**0.12**	0.00	0.00	0.29	**0.39**
C2	0.00	0.00	0.07	**0.20**	0.01	0.00	0.06	**0.16**

Table 14: Performance of LoRA fine-tuned LLaMA-3-Instruct 7B model. Evaluations of DemoGen on GSRR and ReaSCAN do not omit correct outputs for the query from the supports for fairer comparison with CR and GR, because there can exist examples requiring either the same reasoning or outputs in the training data, which artificially boosts the performance of CR and GR. In this situation, the demonstration generator still needs to generate the correct support output, it just isn't omitted from the supports if it happens to be generated.

"not relevant" since they concern an object with different properties than the one mentioned in the query. Examples in red are not correctly executed. Examples in grey are not valid in the environment. Note that for retrieval-based methods like **GandR** and **Retrieval**, the instruction is being solved in a different state to the query one, which is the reason why the action trajectories are both valid and correct, but look very different from each other. Up to 9 of the 16 possible supports are shown.

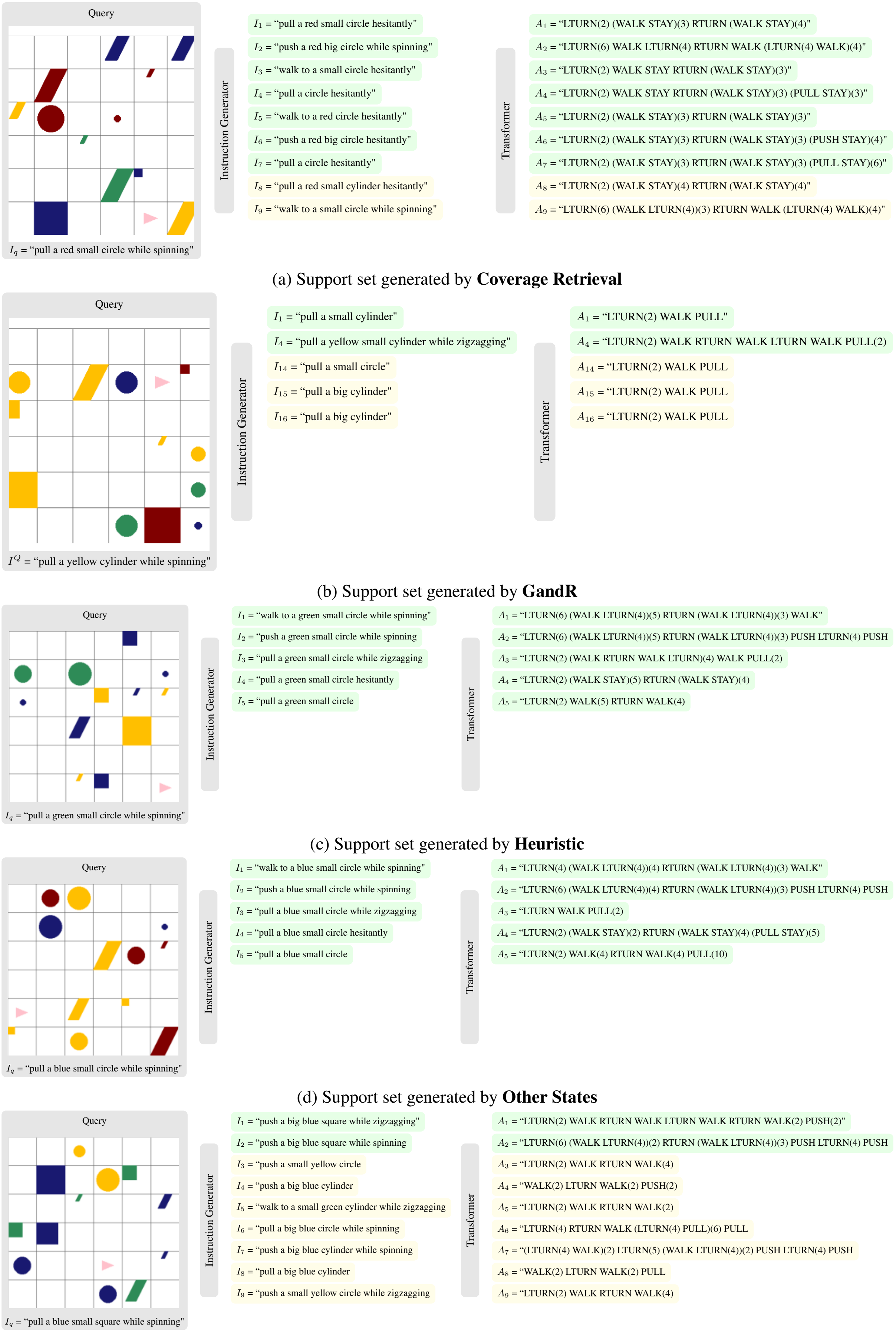
Notice that **GandR** does not demonstrate the desired adverb "while spinning" (`WALK(4)`), because it is only finding near neighbours of "pull", which happen only with `WALK` and `PUSH`.

Relevance	Split	Match % ± std. (N)	Diversity	Split	Match % ± std. (N)
0.49	c	0.67 ± .48 (24)	0.65	c	0.48 ± .50 (163)
0.53	c	0.55 ± .50 (209)	0.69	c	0.57 ± .49 (1381)
0.56	c	0.41 ± .49 (749)	0.72	c	0.60 ± .49 (3606)
0.60	c	0.51 ± .50 (1524)	0.75	c	0.71 ± .46 (7857)
0.63	c	0.53 ± .50 (2674)	0.79	c	0.78 ± .42 (9342)
0.66	c	0.52 ± .50 (4093)	0.82	c	0.67 ± .47 (5855)
0.70	c	0.61 ± .49 (6931)	0.86	c	0.53 ± .50 (4777)
0.73	c	0.66 ± .47 (8906)	0.89	c	0.53 ± .50 (3049)
0.76	c	0.79 ± .41 (11385)	0.92	c	0.56 ± .50 (1261)
0.80	c	0.89 ± .31 (925)	0.95	c	0.63 ± .49 (129)
0.61	h	0.57 ± .50 (60)	0.61	h	0.86 ± .34 (266)
0.64	h	0.59 ± .49 (743)	0.65	h	0.87 ± .33 (2655)
0.68	h	0.59 ± .49 (2907)	0.69	h	0.88 ± .32 (8144)
0.71	h	0.63 ± .48 (4412)	0.73	h	0.86 ± .35 (7480)
0.74	h	0.80 ± .40 (6025)	0.77	h	0.78 ± .41 (7283)
0.78	h	0.81 ± .39 (10824)	0.81	h	0.76 ± .43 (5358)
0.81	h	0.86 ± .35 (10530)	0.85	h	0.61 ± .49 (3910)
0.84	h	0.87 ± .33 (3071)	0.89	h	0.60 ± .49 (1738)

Table 15: Diversity Relevance score bin lower bounds and exact match performance on the gSCAN DemoGen dataset, split C and H.

Dataset	Example
gSCAN	State: agent d: 1 x: 3 y: 5, blue box s: 1 x: 5 y: 5, blue box s: 3 x: 3 y: 4, yellow cylinder s: 1 x: 5 y: 4, yellow cylinder s: 3 x: 5 y: 2, yellow box s: 3 x: 2 y: 3, yellow box s: 4 x: 0 y: 3, green cylinder s: 4 x: 3 y: 2, green cylinder s: 1 x: 2 y: 1, red circle s: 2 x: 3 y: 3, red circle s: 3 x: 2 y: 5, green box s: 2 x: 0 y: 4, green box s: 1 x: 4 y: 2 Query Input: walk to a yellow small square hesitantly Output: **lturn lturn walk stay walk stay walk stay walk stay rturn walk stay walk stay walk stay walk stay walk stay [eos]**
ICL gSCAN	Complete based on the following. Base the answer on Inputs Output pairs that are relevant to the Query Input: Input: walk to a small circle Output: lturn walk walk walk walk walk [eos] Input: pull a small circle Output: lturn walk walk walk walk walk [eos] Input: push a small square hesitantly Output: lturn walk stay walk stay push stay [eos] Input: push a small circle while spinning Output: lturn lturn lturn lturn lturn walk lturn lturn lturn lturn walk lturn lturn lturn lturn walk lturn lturn lturn lturn walk lturn lturn lturn lturn walk [eos] Input: push a small circle while zigzagging Output: lturn walk walk walk walk walk [eos] Input: push a yellow small circle Output: lturn lturn walk walk walk rturn walk walk walk walk walk [eos] Input: push a yellow small circle hesitantly Output: lturn lturn walk stay walk stay walk stay rturn walk stay walk stay walk stay walk stay walk stay [eos] Input: push a small circle Output: lturn walk walk walk walk walk [eos] Query Input: push a small circle hesitantly Output: **lturn walk stay walk stay walk stay walk stay walk stay [eos]**
ICL ReaSCAN	Complete based on the following. Base the answer on Inputs Output pairs that are relevant to the Query Input: State: agent d: 1 x: 4 y: 0, yellow cylinder s: 3 x: 0 y: 2, blue circle s: 1 x: 0 y: 0, yellow square s: 4 x: 1 y: 2, blue cylinder s: 3 x: 1 y: 3, green square s: 4 x: 0 y: 3, red cylinder s: 3 x: 0 y: 4, blue circle s: 1 x: 0 y: 1, yellow cylinder s: 1 x: 2 y: 5, yellow circle s: 3 x: 2 y: 0, green square s: 4 x: 0 y: 5, blue circle s: 4 x: 5 y: 1, green square s: 1 x: 4 y: 3, yellow cylinder s: 2 x: 3 y: 1, yellow circle s: 2 x: 3 y: 0, blue square s: 4 x: 1 y: 1 Input: pull the cylinder that is in the same row as a small yellow square and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk pull pull pull pull [eos] Input: pull the object that is in the same row as a small yellow square and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk walk pull pull pull pull [eos] Input: pull the object that is in the same row as a small yellow circle and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk pull pull pull pull [eos] Input: pull the small red object that is in the same row as a small yellow square and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk pull pull pull pull pull pull pull pull [eos] Input: pull the big red cylinder that is in the same row as a small yellow square and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk pull pull pull pull pull pull [eos] Input: pull the big red cylinder that is in the same row as a small yellow circle and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk walk pull pull pull pull pull pull pull pull [eos] Input: pull the cylinder that is in the same row as a small yellow circle and in the same column as a big green circle while zigzagging Output: walk lturn walk rturn walk lturn walk rturn walk lturn walk pull pull pull pull [eos] Input: pull the Output: walk walk walk pull pull pull [eos] Query Input: pull the cylinder that is in the same row as a small yellow square and in the same column as a big circle while zigzagging Output: **walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk lturn walk rturn walk [eos]**
GSRR	State: agent d: 1 x: 1 y: 1, green cylinder s: 3 x: 4 y: 5, red square s: 2 x: 5 y: 4, green cylinder s: 3 x: 1 y: 2, green cylinder s: 3 x: 4 y: 2, yellow square s: 2 x: 0 y: 1, green cylinder s: 2 x: 5 y: 1, yellow cylinder s: 4 x: 2 y: 1, yellow cylinder s: 2 x: 5 y: 0, green square s: 3 x: 3 y: 3, green square s: 3 x: 3 y: 1, yellow square s: 2 x: 3 y: 2, yellow square s: 4 x: 4 y: 0, green circle s: 1 x: 3 y: 0, green circle s: 2 x: 0 y: 3, blue circle s: 4 x: 2 y: 2, blue circle s: 4 x: 5 y: 2 Query Input: push a green big cylinder north east of a blue circle, Output: **walk walk walk walk walk rturn walk walk [eos]**

Figure 9: Examples of inputs to language-model for evaluation on an LLM. Bolded text is generated from the LLM autoregressively.



I_q = “pull a red small circle while spinning"

I_1 = “pull a red small circle hesitantly"

I_2 = “push a red big circle while spinning"

I_3 = “walk to a small circle hesitantly"

I_4 = “pull a circle hesitantly"

I_5 = “walk to a red circle hesitantly"

I_6 = “push a red big circle hesitantly"

I_7 = “pull a circle hesitantly"

I_8 = “pull a red small cylinder hesitantly"

I_9 = “walk to a small circle while spinning"

A_1 = “LTURN(2) (WALK STAY)(3) RTURN (WALK STAY)(4)"

A_2 = “LTURN(6) WALK LTURN(4) RTURN WALK (LTURN(4) WALK)(4)"

A_3 = “LTURN(2) WALK STAY RTURN (WALK STAY)(3)"

A_4 = “LTURN(2) WALK STAY RTURN (WALK STAY)(3) (PULL STAY)(3)"

A_5 = “LTURN(2) (WALK STAY)(3) RTURN (WALK STAY)(3)"

A_6 = “LTURN(2) (WALK STAY)(3) RTURN (WALK STAY)(3) (PUSH STAY)(4)"

A_7 = “LTURN(2) (WALK STAY)(3) RTURN (WALK STAY)(3) (PULL STAY)(6)"

A_8 = “LTURN(2) (WALK STAY)(4) RTURN (WALK STAY)(4)"

A_9 = “LTURN(6) (WALK LTURN(4))(3) RTURN WALK (LTURN(4) WALK)(4)"

(a) Support set generated by **Coverage Retrieval**

I^Q = “pull a yellow cylinder while spinning"

I_1 = “pull a small cylinder"

I_4 = “pull a yellow small cylinder while zigzagging"

I_{14} = “pull a small circle"

I_{15} = “pull a big cylinder"

I_{16} = “pull a big cylinder"

A_1 = “LTURN(2) WALK PULL"

A_4 = “LTURN(2) WALK RTURN WALK LTURN WALK PULL(2)

A_{14} = “LTURN(2) WALK PULL

A_{15} = “LTURN(2) WALK PULL

A_{16} = “LTURN(2) WALK PULL

(b) Support set generated by **GandR**

I_q = “pull a green small circle while spinning"

I_1 = “walk to a green small circle while spinning"

I_2 = “push a green small circle while spinning

I_3 = “pull a green small circle while zigzagging

I_4 = “pull a green small circle hesitantly

I_5 = “pull a green small circle

A_1 = “LTURN(6) (WALK LTURN(4))(5) RTURN (WALK LTURN(4))(3) WALK"

A_2 = “LTURN(6) (WALK LTURN(4))(5) RTURN (WALK LTURN(4))(3) PUSH LTURN(4) PUSH

A_3 = “LTURN(2) (WALK RTURN WALK LTURN)(4) WALK PULL(2)

A_4 = “LTURN(2) (WALK STAY)(5) RTURN (WALK STAY)(4)

A_5 = “LTURN(2) WALK(5) RTURN WALK(4)

(c) Support set generated by **Heuristic**

I_q = “pull a blue small circle while spinning"

I_1 = “walk to a blue small circle while spinning"

I_2 = “push a blue small circle while spinning

I_3 = “pull a blue small circle while zigzagging

I_4 = “pull a blue small circle hesitantly

I_5 = “pull a blue small circle

A_1 = “LTURN(4) (WALK LTURN(4))(4) RTURN (WALK LTURN(4))(3) WALK"

A_2 = “LTURN(6) (WALK LTURN(4))(4) RTURN (WALK LTURN(4))(3) PUSH LTURN(4) PUSH

A_3 = “LTURN WALK PULL(2)

A_4 = “LTURN(2) (WALK STAY)(2) RTURN (WALK STAY)(4) (PULL STAY)(5)

A_5 = “LTURN(2) WALK(4) RTURN WALK(4) PULL(10)

(d) Support set generated by **Other States**

I_q = “pull a blue small square while spinning"

I_1 = “push a big blue square while zigzagging"

I_2 = “push a big blue square while spinning

I_3 = “push a small yellow circle

I_4 = “push a big blue cylinder

I_5 = “walk to a small green cylinder while zigzagging

I_6 = “pull a big blue circle while spinning

I_7 = “push a big blue cylinder while spinning

I_8 = “pull a big blue cylinder

I_9 = “push a small yellow circle while zigzagging

A_1 = “LTURN(2) WALK RTURN WALK LTURN WALK RTURN WALK(2) PUSH(2)"

A_2 = “LTURN(6) (WALK LTURN(4))(2) RTURN (WALK LTURN(4))(3) PUSH LTURN(4) PUSH

A_3 = “LTURN(2) WALK RTURN WALK(4)

A_4 = “WALK(2) LTURN WALK(2) PUSH(2)

A_5 = “LTURN(2) WALK RTURN WALK(2)

A_6 = “LTURN(4) RTURN WALK (LTURN(4) PULL)(6) PULL

A_7 = “(LTURN(4) WALK)(2) LTURN(5) (WALK LTURN(4))(2) PUSH LTURN(4) PUSH

A_8 = “WALK(2) LTURN WALK(2) PULL

A_9 = “LTURN(2) WALK RTURN WALK(4)

(e) Support set generated by **Random Instructions**

Figure 10: Demonstrations generated on Split H for different kinds of demonstration strategies.