UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Re] A Replication Study of Compositional Generalization Works on Semantic Parsing

Kaiser Sun[1], Adina Williams[1,2], and Dieuwke Hupkes[1,2]

[1]Meta AI, 1601 Willow Road Menlo Park, CA 94025, United States – [2]Shared senior authorship

Edited by
(Editor)

Received
01 November 2018

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility – We examine the reproducibility of compositional generalization results from the task of semantic parsing. We aim to reproduce results from [1], [2], and [3] and seek to verify the claims that 1. A model shouldn't be expected to perform well on non-synthetic datasets just because it performs well on SCAN [1], 2. The approaches from [1] and [2] meet or exceed baseline performance on compositional generalization tests, and 3. NQG-T5 [1] outperforms baselines on both synthetic and natural data. 4. NQG [1] performs well on the instances that it is able to generate a prediction, but it faces the barrier of not being able to generate predictions for all instances.

Methodology – We reuse the authors' code along with additional code to run extra experiments, and we re-implement scripts whose support is deprecated. Eight 32GB GPUs were used for experiments, with a detailed description in Section 3.3.

Results – Claim 1 is verified: the model with the highest performance on SCAN does not maintain its high performance on other datasets (Section 4.1). Claims 2 and 3 are verified, with a comparison of performance between NQG-T5 and the selected baseline models in [1] and [2]. Claim 4 is also verified by computing the coverage and precision of NQG in Section 4.4. Overall, accuracy for most experiments reaches within 2% of that reported in the original paper, with a deviation that our T5 achieves higher performance on some splits and slightly lower performance on one split than reported previously.

What was easy – All papers provide clearly-written code and informative documentation, as well as lists of hyperparameters that are used for experiments. The papers also describe their approaches clearly, making the experimental workflow easy to follow.

What was difficult – The exact match evaluation metric is formulated somewhat differently across all three papers, leading to non-negligible value differences, as discussed in Section 5.2. We also had to re-implement some training scripts because an original dependency is no longer supported. Moreover, some experiments are computationally expensive: [1] used TPUs for experiments, while our replication with GPUs takes several days to train a single T5 model.

Correspondence should be addressed to Kaiser Sun (kaisersun@meta.com)
The authors have declared that no competing interests exist.
Code is available at https://github.com/KaiserWhoLearns/CompGenRep_MLRC2022. – SWH swh:1:dir:f508ee8f31bd7a768d1fa09e7fedf834e663fd6e.
Open peer review is available at https://openreview.net/forum?id=MF9uv95psps.

Communication with original authors — The authors of all three papers provided us with useful instruction to work with their methods and constructive feedback on the draft.

1 Introduction

In this work, we reproduce and connect results from three papers [1, 2, 3] that focus on compositional generalization in semantic parsing. More specifically, we train and evaluate four models – LSTM [4], T5 [5], Neural-QCFG [2], and NQG [1] – on different splits of synthetic and realistic datasets – SCAN [6], GEOQUERY, SPIDER [7], and COGS [3] – for evaluating compositional generalization. Broadly speaking, we are interested in whether model performance on synthetic datasets aligns with performance on more realistic datasets.[1] To this end, we aim to replicate [1], who find that performance of several pre-trained models on SCAN, a synthetic dataset, does not align with performance on GEOQUERY and SPIDER, whose instances are crafted by humans. We then extend their findings in two ways. On the data side, to increase the variation, we add experiments with an additional synthetic compositional generalisation dataset: COGS [3]. On the modelling side, on the other hand, we add two additional model architectures – a vanilla LSTM (as used by [3]) and the sequence-to-sequence (seq2seq) model Neural-QCFG proposed by [2], which incorporates parametrized grammars to capture hierarchical structure. In doing so, we replicate also (part of) the results from [2] and [3].

2 Scope of reproducibility

In this work, we aim to verify the following claims:

- Claim 1. For T5 and NQG-T5, high performance on SCAN does not entail high performance on non-synthetic datasets [1].
- Claim 2. NQG-T5 [1] and Neural-QCFG [2] match or exceed the accuracy of baselines for compositional generalization.
- Claim 3. Compared to the baseline models that excel at synthetic datasets, NQG-T5 performs better on both synthetic and natural data.
- Claim 4. NQG can have issues generating predictions for some specific instances due to the limitation of grammar induction. But if NQG is able to generate predictions for test instances, it performs well on these instances.

Claim 1 is verified by our evaluations of T5 and NQG-T5 on SCAN, GEOQUERY, and SPIDER. We extend Claim 1 by additionally measuring performance on COGS, a synthetic dataset proposed by [3], aiming to assess whether the performance difference is specific to SCAN in particular or holds for synthetic datasets more generally (§4.1). For Claim 2, we reproduce the proposed models and compare them with the baseline models from [1] and [2] (§4.2). We also specifically explore the performance of NQG-T5 on both synthetic and realistic datasets overall to verify Claim 3 (§4.3). Finally, we compute the precision and coverage of our NQG model in §4.4.

3 Methodology

We maximize our reuse of the code from the authors of the original papers: we reuse all the code from [2] and [3]; For [1], we reuse the code for NQG, and refactor the code for fine-tuning T5 into PyTorch with Huggingface Transformers,[2] because the original T5 dependency [3] is no longer maintained, and PyTorch aligns better with the dependencies in the other two studies. We address minor issues caused by versioning or sequence

[1] Following [1], we deem a dataset *synthetic* if its instances are heuristically generated by a program, we deem a dataset *natural/realistic* if its instances are produced directly by humans.

[2] https://github.com/huggingface/transformers

[3] https://github.com/google-research/text-to-text-transfer-transformer

	Type	LSTM	T5	NQG	Neural-QCFG
SCAN	Synthetic	Us	[1]	[1]	[2]
COGS		[3]	Us	-	-
GEOQUERY	Realistic	Us	[1]	[1]	-
SPIDER		Us	[1]	[1]	-

Table 1. A list of datasets and models used in the papers. The type column indicates whether the dataset is synthetic or realistic. Us denotes additional experiments conducted by us.

Dataset	Split	Train	Validation	Test	Overall
GEOQUERY	Standard	600	-	280	880
	Length	440	-	440	880
	TMCD	440	-	440	880
	Template	441	-	439	880
SPIDER	Random	3,282	-	1,094	4,376
	Length	3,282	-	1,094	4,376
	TMCD	3,282	-	1,094	4,376
	Template	3,280	-	1,096	4,376
SCAN	Standard	16,990	-	3,920	20,910
	Length	16,990	-	3,920	20,910
	Add Prmitive (JUMP)	14,670	-	7,706	22,376
	Add Primitive (TURN LEFT)	21,890	-	1,208	23,098
	Template (AROUND RIGHT)	15,225	-	4,476	19,701
	MCD1	8,365	1,045	1,045	10,455
	MCD2	8,365	1,045	1,045	10,455
	MCD3	8,365	1,045	1,045	10,455
COGS	Standard	24,155	3,000	3,000	30,155
	Generalization	-	-	21,000	21,000

Table 2. Number of instances for each dataset in each optimization split.

truncation during tokenization for each repository. Finally, we refactor the code for [1], [2], and [3] into a one-stop repository with a cleaned-up dependency and unified experimental scripts. For experiments on T5, we use eight Tesla V100 GPUs with 32GB CUDA memory each, and a single V100 or 16GB Quadro GP100 for the rest of the models. Section 3.3 includes a detailed list of computational resources we use. We list the datasets and the models below, with a summary of the experiments in Table 1.

3.1 Datasets

The datasets we consider can be classified as either synthetic or realistic. Previous work on compositional generalization [8, 9, 10, 11, 12, 13, 14] focused on modeling approaches that excel on synthetic datasets such as SCAN [15], while [1] is motivated by the question of whether semantic parsing approaches can handle both synthetic and realistic data. Each dataset we consider is divided into a training and a test set according to a different splitting strategy. For *random* or *standard* splits, the instances are assigned randomly to either the training or test set. For *template* splits, instances that satisfy specific pattern will be isolated from the training set and can only appear in the test set. In *length* splits, the instances with longer output (query length for SPIDER and GEOQUERY, command sequence length for SCAN) are allocated to the test set, and the remaining shorter instances comprise the training set. Maximum Compound Divergence (*MCD*) is a splitting strategy introduced by [16] that maximizes compound divergence at a low atom divergence between train and test set. *MCD* requires that both source and target be generated by a rule-based method, thus [1] propose Target Maximum Compound Divergence (*TMCD*) splits, which is comparable to *MCD* but is also applicable to realistic datasets. In [1], the *MCD* approach is applied on SCAN, while *TMCD* is applied on GEOQUERY

and SPIDER. Below are descriptive details for each dataset and their splits. Appendix 7 includes example instances from the datasets.

COGS. COGS is a **synthetic** semantic parsing dataset created for assessing compositional generalization [3]. The inputs are English sentences, generated by a Probabilistic Context-Free Grammar (PCFG). The corresponding output, which is the semantic interpretation of the input, is annotated with the logical formalism of Reddy et al.[17] and enhanced with a couple of postprocessing procedures. COGS provides four different sets: *train, development, test*, and *generalization* sets. The instances in the generalization set are created from separate PCFGs, while the other three contain instances constructed with the same PCFGs. Unlike the other datasets we used, COGS does not introduce additional splits beyond the *generalization* split.

SCAN. SCAN is a **synthetic** dataset in which English commands are to be converted into sequences of prespecified actions. The actions are composed of simple movement designations such as "JUMP" or "TURN RIGHT". In addition to the *random, length, template,* and *MCD* split introduced above, two additional splits of SCAN from [15] are used in [1] and [2]:

- *Add primitive (JUMP)* - The training set excludes the commands with the primitive "JUMP"; the test set includes compositional commands that use it.
- *Add primitive (TURN LEFT)* - Similar to the prior split, this splitting method isolates all the "TURN LEFT" commands in the training set.

GEOQUERY. GEOQUERY [18, 19] contains natural language questions about US geography. A model is fed the question and is expected to output the corresponding query, which can be executed to search a database. [1] convert all entity mentions with placeholders and used a variant, Functional Query Language (FunQL), as the target representation. In [1], four splits are constructed: *standard, template, length,* and *TMCD*, and are used for training the models.

SPIDER. SPIDER is a text-to-SQL dataset that spans multiple domains. SPIDER is originally designed for corss-domain semantic parsing and incorporates challenges to generalize to new database schemas by using different database in training and test set. It also possesses a more complex syntax of SQL. [1] adopt a setting where databases are shared between train and test examples, so that the dataset splits can be dedicated to evaluating compositional generalization. Similar to GEOQUERY, the following splits are generated in [1]: *standard, template, length,* and *TMCD*.

3.2 Models

NQG. To work towards a semantic parsing approach that can handle both compositional generalization and natural language variation, Shaw et al.[1] proposed an ensemble, NQG-T5, that chains a grammar-based model with a pre-trained seq2seq model. The grammar-based component, made up of a **N**eural parsing model with **Q**uasi-synchronous **G**rammar induction, first induces a QCFG, then trains a discriminative latent variable parsing model to make derivations with the induced grammar. On instances for which NQG cannot provide a output, Shaw et al.[1] fall back on T5 [5] to make predictions.

Neural-QCFG. Kim[2] also use a quasi-synchronous grammar in the proposed approach, Neural-Quasy-Synchronous-Grammar QCFG (Neural-QCFG). In contrast to [1], Neural-QCFG parameterizes the grammar's rule probabilities and treat the source and target trees as latent variables during training, and has no fall-back module. Therefore, Neural-QCFG performs end-to-end generation and is easily applicable to a wide range of seq2seq tasks.

T5. T5 [5] is an encoder-decoder Transformer [20] model that is pre-trained on multiple tasks. Each task is converted into a seq2seq format with a task-specific prefix, thus making it generally applicable to a variety of tasks. [1] use T5-base and T5-3B as both a baseline and a fallback model when NQG fails to produce a target. Due to computational constraints, we will only be reproducing the results with T5-base.

LSTM. Long Short-term Memory (LSTM) is a classical neural network that is widely used for a substantial amount of tasks. Because it does not contain any pre-trained knowledge, LSTM is an ideal candidate to provide a sense of how classical models perform on the compositional generalization datasets. Kim and Linzen[3] train uni- and bi-LSTM on COGs, and we are able to reproduce their results of LSTM on COGS, and also to train LSTM on other datasets used in [1] and [2] for comparison.

3.3 Experimental Setup

We train/fine-tune the models on each dataset as specified above, and evaluate on the corresponding test set.[4] For COGS, we also evaluate models on the generalization set. We use exact-match accuracy (EM) as evaluation metric. Note that because the vocabulary of T5 does not contain the "<" symbol, which appears in a large amount of instances, all UNK tokens in the output of T5 are considered as "<" during evaluation. For LSTM, we use five different random seeds and evaluate the averaged performance.

Hyperparameters. Following the original authors, we use a learning rate of 1.0×10^{-4} and an equivalent batch size of 256 for experiments with T5. We fine-tune for 2,400 steps on GEOQUERY and 10,000 steps on SPIDER. Because we do not have access to TPUs that were used in [1], we add in a gradient accumulation step of 16 to achieve the same equivalent batch size. We only optimize for 2,400 steps instead of 3,000 steps in GEOQUERY because no clear improvement in performance is observed after 1,000 steps. [1] reported T5 results on SCAN from [16], who used a different set of hyperparameters. In our experiments for T5 on SCAN, we conducted a minimal hyperparameter searching among setups from [16] and [1], with a commonly used learning rate of 1.0×10^{-3}, and arrive at a better performance with the hyperparameters from [1]. Therefore, all the results on T5 in the next section follow the hyperparameters of [1], with an optimization steps of 4,550, the step size that we start to observe convergence.
For experiments with NQG, we use the original set up, except that we use BERT-Base model for SCAN and SPIDER, as opposed to [1]'s BERT-Tiny model, because the original BERT-Tiny model is no longer available in the Tensorflow model release[5] used by the original paper. For each trial, NQG is fine-tuned for 256 steps with a learning rate of 1.0×10^{-4} and equivalent batch size of 256. For Neural-QCFG, we employed the same set of hyperparameters as [2], which includes an Adam optimizer [21] with learning rate of 5.0×10^{-4}, gradient norm clipping at 3, and a L_2 penalty of 10^{-5}. With a batch size of 4, the model is trained for 15 epochs with early stopping on best performing checkpoint on validation set. For LSTMs, we adopt the hyperparameters from [3], which use a Noam learning rate scheduler [20], with an initial learning rate of 2 and optimize for 30,000 steps. The equivalent batch size for LSTMs is 128.

Computational Requirements. Eight Tesla V100 GPUs, each with 32GB memory, are used for the experiments with T5. For experiments with smaller models such as Neural-QCFG and NQG, one V100 is used. A single 16GB Quadro GP100 is used for training the LSTM. We report the average GPU hours spent for training models on each dataset in Table 8 in the Appendix.

[4] Our replication code can be found in https://anonymous.4open.science/r/CompGen/; all code will be made public upon acceptance.
[5] https://github.com/tensorflow/models

Model		LSTM-Uni		LSTM-Bi		T5		NQG		NQG-T5		Neural-QCFG	
Dataset	Split	EM	Orig	EM	Orig	EM	Orig	EM	Orig	EM	Orig	EM	Orig
COGS	Test	99.0	0.0	99.0	99.0	98.7	-		-		-		-
	Gen	32.0	32.0	23.0	16.0	80.7	-		-		-		-
SCAN	Rand.	13.3	-	14.5	-	77.8	-	100.0	100.0	100.0	100.0	96.1	96.9
	Length	15.3	-	11.8	-	13.6	14.4	100.0	100.0	100.0	100.0	91.6	95.7
	Jump	0.4	-	0.0	-	93.5	99.5	100.0	100.0	100.0	100.0	94.3	96.8
	Turn L.	61.1	-	34.1	-	61.9	62.0	100.0	100.0	100.0	100.0	76.2	-
	Temp.	0.2	-	0.3	-	37.6	-	0.0	-	0.0	-	96.9	98.7
	MCD	7.1	-	8.6	-	23.3	15.4	100.0	100.0	100.0	100.0	-	-
GEOQ.	Stan.	72.8	-	80.1	-	92.9	92.9	72.5	76.3	90.7	92.9		-
	Length	17.3	-	15.8	-	48.0	39.1	25.7	37.4	46.6	52.2		-
	Temp.	46.5	-	55.9	-	91.3	87.0	59.2	61.9	85.6	88.8		-
	TMCD	35.8	-	37.1	-	54.1	54.3	39.1	41.1	50.2	56.6		-
SPIDER	Rand.	33.6	-	37.3	-	77.5	76.5	0.0	1.3	79.1	81.8		-
	Length	12.7	-	14.1	-	44.3	42.5	0.0	0.0	44.3	49.0		-
	Temp.	1.3	-	2.3	-	53.2	45.3	0.0	0.5	53.2	59.2		-
	TMCD	4.7	-	6.2	-	57.0	42.3	0.0	0.5	57.0	60.8		-

Table 3. Model performance on each dataset, evaluated by exact-match accuracy (EM). 'Orig' represents the EM values reported by the original paper.

4 Results

Table 3 shows replicated and original model performance on all datasets. Overall, the metric values we obtained are close to the values originally reported (within 5%), with the exception of T5: on the *template* and *TMCD* split of SPIDER and on the *length* split of GEOQUERY, T5 achieves a much higher EM than was reported in the original paper [1]; and it obtains a 6% lower EM on the *AddPrimitive_Jump* split of SCAN.

4.1 Result 1: High performance on SCAN does not entail high performance on non-synthetic datasets.

NQG performs the best on all splits of SCAN with one exception on the *template* split, see Table 3. However, it fails on SPIDER, and T5 outperforms it on GEOQUERY. The T5 results align with Claim 1: T5 performs well on SPIDER and GEOQUERY, but achieves scores as low as 13.6% on some splits of SCAN. NQG-T5, while it performs almost perfectly on SCAN and achieves a respectable performance on SPIDER, is an ensemble whose performance on SCAN comes from its NQG module, whereas its performance on SPIDER comes from its T5 module, thus results on NQG-T5 cannot lead us to reject Claim 1. We have thus verified that the performance on GEOQUERY and SPIDER is not predicable from the performance on SCAN. It is worth noting, however, that even within the same dataset creation method, the model performance can be very different: our LSTMs perform well on the *random* split of GEOQUERY, but perform poorly on *random* split of SPIDER; both datasets are natural datasets. In addition, T5 performs well on both the training and generalization set of COGS, but not on splits of SCAN; both SCAN and COGS are synthetic datasets. NQG also performs poorly on the *template* split of SCAN, whose other splits NQG excels at. Overall, not only is it hard to predict anything based on model performance on SCAN, it appears to be difficult to predict the performance on any dataset, given the performance on another in our list.

4.2 Result 2: NQG-T5 and Neural-QCFG perform as well as or better than the baseline approaches for compositional generalization

As an ensemble of NQG and T5, NQG-T5 undoubtedly achieves 100% EM in most splits of SCAN. It also performs better on SPIDER and GEOQUERY than the baseline models in Table 5). However, for splits of GEOQUERY, the performance of T5 exceeds that of

Approach	Stan.	Temp.	Len.	TMCD
Syn Attn [22]	77.5	70.6	23.6	0.0
CGPS [23]	62.1	32.8	9.3	32.3
T5-base	92.9	91.1	48.0	53.9
NQG	72.5	59.2	25.7	39.1
NQG-T5	90.7	85.6	46.6	50.2

Table 4. Performance of NQG-T5 and its baselines on the splits of GEOQUERY. Stan., Temp., and Len. are abbreviations of standard, template, and length respectively.

Approch	Simple	Jump	Right	Length
RNN [15]	99.7	1.7	2.5	13.8
CNN [24]	100.0	69.2	56.7	0.0
Syn Attn [22]	100.0	91.0	28.9	15.2
CGPS [23]	99.9	98.8	83.2	20.3
Eq. Seq2Seq [11]	100.0	99.1	92	15.9
LANE [12]	100.0	100.0	100.0	100.0
Program Syn. [14]	100.0	100.0	100.0	100.0
NeSS [13]	100.0	100.0	100.0	100.0
NQG-T5	100.0	100.0	-	100.0
T5-Base	77.8	93.5	33.2	13.6
Neural QCFG	96.1	94.3	96.85	91.6

Table 5. Performance of NeuralQCFG on splits of SCAN. The baselines except for NQG-T5, T5-base are not reproduced by us and are cited from [2].

	Standard		Length		Template		TMCD	
	Ours	Orig	Ours	Orig	Ours	Orig	Ours	Orig
Precision	95.8	95.7	81.0	86.4	88.4	95.8	95.0	94.1
Coverage	75.7	80.2	33.4	43.3	67.0	64.5	41.1	43.7
	Jump		Turn L.		Length		MCD	
	Ours	Orig	Ours	Orig	Ours	Orig	Ours	Orig
Precision	100	100	100	100	100	100	100	100
Coverage	100	100	100	100	100	100	100	100

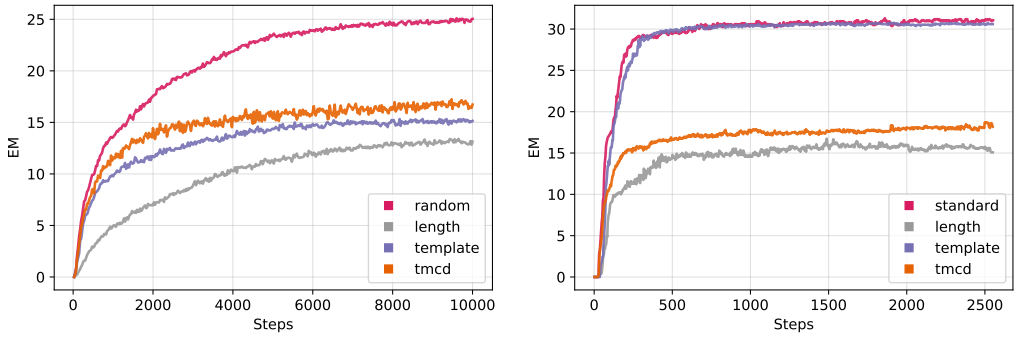
Table 6. Precision and coverage of NQG on splits of GEOQUERY (top) and SCAN (bottom).

NQG-T5. This is because NQG-T5 will only fallback to T5's prediction if NQG is unable able to generate one, suggesting that NQG either makes more mistakes on these splits or covers most instances that T5 predicts correctly. We will explore a possible reason for this in §4.4. Therefore, NQG-T5 outperforms all baselines other than T5, and it performs comparably to T5. The claim that NQG-T5 performs better or comparably to baseline approaches holds.

Neural-QCFG also shows a much higher performance than T5 on SCAN, although it is not able to achieve an EM as high as NQG-T5's.[6] Compared to the baseline approaches listed in Table 5, Neural-QCFG performs stably across splits. Although not beating other approaches that achieve nearly perfect performance, Neural-QCFG does not rely on SCAN-specific information, which might enable a more straight-forward application to natural domain, as explained by Kim[2].[7]

4.3 Result 3: Compared to the models that excel at synthetic datasets, NQG-T5 outperforms them in both synthetic and natural data.

Table 4 shows NQG-T5 performance against the baseline models that are also evaluated on GEOQUERY. Although NQG-T5 is surpassed by T5 on splits of GEOQUERY by a slight amount, it outperforms the other baselines drastically on all datasets, with a perfect performance on SCAN (Table 5). Therefore, we are able to verify Claim 3 that NQG-T5 performs well on both synthetic and natural datasets compared to the baseline models, with a caveat of performing slightly worse than T5 on GEOQUERY.



(a) Training Curve of T5 on SPIDER **(b)** Training Curve of T5 on GEOQUERY

Figure 1. Training curve of T5-base on different splits of SPIDER and GEOQUERY.

4.4 Result 4: NQG presents high precision but struggles at coverage on some data splits.

Following Shaw et al.[1], we compute the coverage (the percentage of examples where NQG is able to generate a prediction) and the precision (the accuracy of NQG among the instances that it can produce a prediction) in Table 6. Predictably from its overall performance, NQG covers all instances in SCAN and achieves 100% precision. For GEOQUERY, all the numbers are within 5% of those originally reported, except for the coverage of length split on GEOQUERY. Like the original authors, we observe that the precision of our NQG is higher than its overall EM as shown in Table 3, suggesting that NQG struggles to cover more instances and it generally performs well if it can produce a prediction on that instance. Hence, the claim that NQG presents high precision but struggles with coverage holds on the splits of GEOQUERY; NQG already achieves high precision and high coverage on SCAN.

4.5 Result 5: T5 converges early on SPIDER and GEOQUERY, suggesting early stopping might help save compute resources.

While fine-tuning T5 on SPIDER and GEOQUERY, which is optimized for 10,000 steps and 2,400 steps respectively, we find that the model reaches a similar performance to originally reported at checkpoints earlier than those originally reported (Figure 1). T5 approximates final performance as early as at 7,000 steps in the template split of SPIDER and at 800 steps in multiple splits of GEOQUERY. The training curve implies that the model likely converges substantially earlier than Shaw et al.[1] estimate, thus suggesting that less epochs can be used with a careful early stopping strategy.

5 Discussion

5.1 The ambiguity of selecting a validation set

In §4.5, we observed that T5 converges in approximately the first 70% of the optimization steps. This suggest that employing an early stopping strategy based on the performance on a validation set might help save compute resources. However, the selection of a validation set is far from trivial, especially when there are multiple splits of the same dataset. [1] initially tune hyperparameters on a random held-out set and then further

[6] [2] evaluate Neural-QCFG is also on other compositional generalization tasks, we only present its result on SCAN here because the focus of this work is on semantic parsing.

[7] We intended to extend the experiments by training Neural-QCFG on the natural semantic parsing datasets, but during communication with an author of [2], we understood that a more careful design of data preprocessing to address the special tokens is required for these datasets.

select based on the performance on a validation set. The resulting optimal set of hyperparameters are used for fine-tuning on all the other generalization splits – it is possible that the hyperparameters that the in-domain validation set provides are sub-optimal for generalization sets. Prior work [25] also discusses the necessity of an out-of-domain validation set. In sum, using an appropriate development set may seriously affect the results, but it may be difficult to choose an ideal development set that is suitable for differently distributed test sets.

5.2 The implementation of exact match accuracy (EM)

To reproduce results as closely as possible, we use the implementation of exact-match (EM) accuracy from the respective papers – with the exception of T5, for which we use the Huggingface EM implementation.[8] In doing so, we notice that seemingly small differences across EM implementations caused non-negligible differences in the results. In [1], SPIDER is evaluated with the EM accuracy evaluation script released by [7], which is more tolerant to misplaced spaces; this should not affect the correctness of a prediction. For the other datasets, SCAN and GEOQUERY, the models are evaluated with exact string match (i.e. `output == prediction` in Python). [2] also append the output words with space to enable more tolerant EM computation for SCAN. We find that, if we use exact string match in NQG for T5 in GEOQUERY, T5 has an extremely low EM accuracy, because T5 does not always append spaces before commas as in the original dataset. When we use the more lenient version of EM implementation that disregards the missing space before a comma, we observed an increase in EM as high as 80%.

5.3 Alignment of compositional generalization datasets

All of the datasets and splits we investigate here are designed for assessing compositional generalization, but we find that high performance of some models on these datasets does not entail high performance on the others, even within the same "natural data" type (Table 3). Several confounding factors contribute to these performance differences: for example, NQG does not produce any output on SPIDER, and [1] explained that the SQL in SPIDER has a much more complex grammar than the FunQL in GEOQUERY, resulting in low performance. Moreover, SPIDER instances are generally long, perhaps causing models to struggle more on this dataset than on the others. In this situation, it is difficult to claim that the model generalizes poorly if only the result on one dataset is given. Even if the model is evaluated on multiple datasets, what conclusions about generalizability can we draw when performance varies across datasets, as we find for SCAN, SPIDER, and GEOQUERY? We find it hard to draw any strict conclusion before a careful examination of datasets and confounding factor is conducted. Future work could focus on explicating the differences between these datasets.

5.4 Effort and Contact

What was easy. The authors of all three previous works provide clearly-written code and helpful documentation that made it straight-forward for us to reproduce their results. Lists of hyperparameters are also directly included in their repositories. All three papers also have clearly written methodologies sections that are intuitive to understand. It was slightly more difficult for us to understand how parameterization is done in [2] from the paper text, but the provided code helped clarify the procedure.

What was difficult. It took more time than expected to generate the split in [1], because the datasets used are from different sources. furthermore, the experiments with T5 cost an extensive amount of computing resources, especially for SPIDER. It take several days to train one variant, and there is more than one split that will need to be fine-tuned on.

[8] https://huggingface.co/spaces/evaluate-metric/exact_match

Communication with original authors. We contacted the authors of all three papers for feedback on this draft and received constructive suggestions: we experienced issues while replicating the results of NQG on GEOQUERY, the first author of [1] provided us with detailed instructions for replication and helped with troubleshooting; the first authors of [3] and [2] pointed us to helpful related work and offered feedback on the draft. We thank all the authors for their thoughtful input and timely responses.

6 Conclusion

In this work, we reproduced the results of [1], [2], and [3]. We verified the claim that performance on SCAN does not entail the performance on non-synthetic datasets, and that Neural-QCFG achieves performance comparable with the baseline approaches. We also verified that NQG-T5 outperforms the baselines on both synthetic and natural data. In addition to the results from the paper, we also find that T5 converges early with the training strategy in SPIDER and GEOQUERY. We also highlight that the EM implementations used in the papers are different, and that this has consequences for the results, but we align the EM with each paper to ensure faithful reproduction.

Acknowledgement

We thank Peter Shaw, Yoon Kim, and Najoung Kim for the helpful instructions on reproducing their works and constructive feedback on this work. Thanks also to anonymous reviewers and chairs for their comments and suggestions.

References

1. P. Shaw, M.-W. Chang, P. Pasupat, and K. Toutanova. "Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?" In: **Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)**. Online: Association for Computational Linguistics, Aug. 2021, pp. 922–938. DOI: 10.18653/v1/2021.acl-long.75. URL: https://aclanthology.org/2021.acl-long.75.
2. Y. Kim. "Sequence-to-Sequence Learning with Latent Neural Grammars." In: **Advances in Neural Information Processing Systems**. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. 2021. URL: https://openreview.net/forum?id=pbfAgoc_l2w.
3. N. Kim and T. Linzen. "COGS: A Compositional Generalization Challenge Based on Semantic Interpretation." In: **Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)**. Online: Association for Computational Linguistics, Nov. 2020, pp. 9087–9105. DOI: 10.18653/v1/2020.emnlp-main.731. URL: https://aclanthology.org/2020.emnlp-main.731.
4. S. Hochreiter and J. Schmidhuber. "Long short-term memory." In: **Neural computation** 9.8 (1997), pp. 1735–1780.
5. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: **J. Mach. Learn. Res.** 21.140 (2020), pp. 1–67.
6. B. M. Lake and M. Baroni. "Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks." In: **ICML**. 2018. URL: https://arxiv.org/pdf/1711.00350.pdf.
7. T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, et al. "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task." In: **arXiv preprint arXiv:1809.08887** (2018).
8. Y. Li, L. Zhao, J. Wang, and J. Hestness. "Compositional Generalization for Primitive Substitutions." In: **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4293–4302. DOI: 10.18653/v1/D19-1438. URL: https://aclanthology.org/D19-1438.
9. B. M. Lake. "Compositional generalization through meta sequence-to-sequence learning." In: **Advances in neural information processing systems** 32 (2019).

10. J. Russin, J. Jo, R. O'Reilly, and Y. Bengio. "Compositional generalization in a deep seq2seq model by separating syntax and semantics. apr 2019." In: **URL http://arxiv. org/abs/1904** 9708 ().
11. J. Gordon, D. Lopez-Paz, M. Baroni, and D. Bouchacourt. "Permutation equivariant models for compositional generalization in language." In: **International Conference on Learning Representations**. 2020.
12. Q. Liu, S. An, J.-G. Lou, B. Chen, Z. Lin, Y. Gao, B. Zhou, N. Zheng, and D. Zhang. "Compositional generalization by learning analytical expressions." In: **Advances in Neural Information Processing Systems** 33 (2020), pp. 11416–11427.
13. X. Chen, C. Liang, A. W. Yu, D. Song, and D. Zhou. "Compositional generalization via neural-symbolic stack machines." In: **Advances in Neural Information Processing Systems** 33 (2020), pp. 1690–1701.
14. M. Nye, A. Solar-Lezama, J. Tenenbaum, and B. M. Lake. "Learning compositional rules via neural program synthesis." In: **Advances in Neural Information Processing Systems** 33 (2020), pp. 10832–10842.
15. B. Lake and M. Baroni. "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks." In: **International conference on machine learning**. PMLR. 2018, pp. 2873–2882.
16. D. Keysers et al. "Measuring Compositional Generalization: A Comprehensive Method on Realistic Data." In: **International Conference on Learning Representations**. 2020. URL: https://openreview.net/forum?id=SygcCnNKwr.
17. S. Reddy, O. Täckström, S. Petrov, M. Steedman, and M. Lapata. "Universal Semantic Parsing." In: **Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing**. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 89–101. DOI: 10.18653/v1/D17-1009. URL: https://aclanthology.org/D17-1009.
18. L. R. Tang and R. J. Mooney. "Using multiple clause constructors in inductive logic programming for semantic parsing." In: **European Conference on Machine Learning**. Springer. 2001, pp. 466–477.
19. J. M. Zelle and R. J. Mooney. "Learning to parse database queries using inductive logic programming." In: **Proceedings of the national conference on artificial intelligence**. 1996, pp. 1050–1055.
20. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need." In: **Advances in neural information processing systems** 30 (2017).
21. D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: **3rd International Conference for Learning Representations**. 2015.
22. J. Russin, J. Jo, R. C. O'Reilly, and Y. Bengio. "Compositional generalization in a deep seq2seq model by separating syntax and semantics." In: **arXiv preprint arXiv:1904.09708** (2019).
23. Y. Li, L. Zhao, J. Wang, and J. Hestness. "Compositional generalization for primitive substitutions." In: **arXiv preprint arXiv:1910.02612** (2019).
24. R. Dessì and M. Baroni. "CNNs found to jump around more skillfully than RNNs: Compositional generalization in seq2seq convolutional networks." In: **arXiv preprint arXiv:1905.08527** (2019).
25. R. Csordás, K. Irie, and J. Schmidhuber. "The devil is in the detail: Simple tricks improve systematic generalization of transformers." In: **arXiv preprint arXiv:2108.12284** (2021).
26. D. Hupkes et al. "State-of-the-art generalisation research in NLP: a taxonomy and review." In: **CoRR** abs/2210.03050 (2022). URL: https://doi.org/10.48550/arXiv.2210.03050.

7 Example of Instances

Table 7 shows examples of instances from each dataset.

8 GPU hours per model / dataset

In Table 8, we report the approximate training time in GPU hours for each model and dataset, averaged over splits.

9 GenBench Evaluation Card

To concisely describe our experiments, we add a GenBench evaluation card [26] in Table 9.

COGS	
Input:	Liam hoped that a box was burned by a girl .
Output:	hope . agent (x _ 1 , Liam) AND hope . ccomp (x _ 1 , x _ 6) AND box (x _ 4) AND burn . theme (x _ 6 , x _ 4) AND burn . agent (x _ 6 , x _ 9) AND girl (x _ 9)
SCAN	
Input:	jump opposite left thrice after look opposite right thrice
Output:	I_TURN_RIGHT I_TURN_RIGHT I_LOOK I_TURN_RIGHT I_TURN_RIGHT I_LOOK I_TURN_RIGHT I_TURN_RIGHT I_LOOK I_TURN_LEFT I_TURN_LEFT I_JUMP I_TURN_LEFT I_TURN_LEFT I_JUMP I_TURN_LEFT I_TURN_LEFT I_JUMP
GEOQUERY	
Input:	name all the rivers in m0
Output:	`answer ( intersection ( river , loc_2 ( m0 ) ) )`
SPIDER	
Input:	flight_1: what is the average distance and price for all flights from la? | flight : flno , origin , destination , distance , departure_date , arrival_date , price , aid | aircraft : aid , name , distance | employee : eid , name , salary | certificate : eid , aid
Output:	`select avg(distance) , avg(price) from flight where origin = "los angeles"`

Table 7. Examples of instances in each dataset.

Dataset	T5	N-QCFG	NQG	LSTM
SCAN	880	108	4	4
COGS	-	-	-	1
SPIDER	1720	-	14	4
GEOQUERY	864	-	9	2

Table 8. Approximate training time in GPU hours per model and dataset, averaged over splits.

Motivation					
Practical	*Cognitive* □ △ ◯ ⊙		*Intrinsic*		*Fairness*
Generalisation type					
Compositional □ △ ◯ ⊙	*Structural*	*Cross Task*	*Cross Language*	*Cross Domain*	*Robustness*
Shift type					
Covariate □ △ ◯ ⊙	*Label*		*Full*		*Assumed*
Shift source					
Naturally occuring	*Partitioned natural* □ △		*Generated shift*		*Fully generated* ◯ ⊙
Shift locus					
Train–test □ ◯	*Finetune train–test* △ ⊙		*Pretrain–train*		*Pretrain–test*

Table 9. A GenBench evaluation card [26] that summarizes our experiments. □= Experiments of LSTM and Neural-QCFG on GEOQUERY and SPIDER; △= Experiments of T5 and NQG on GEOQUERY and SPIDER; ◯= Experiments of LSTM and Neural-QCFG on COGS and SCAN; ⊙= Experiments of T5 and NQG on COGS and SCAN.