Improving Compositional Generalization with Latent Structure and Data Augmentation

Linlu Qiu 1*†  Peter Shaw 1*  Panupong Pasupat 1  Pawel Krzysztof Nowak 1  Tal Linzen 1,2  Fei Sha 1  Kristina Toutanova 1

1 Google Research  2 New York University
{luluqiu,petershaw,ppasupat,pawelnow,linzen,fsha,kristout}@google.com

Abstract

Generic unstructured neural networks have been shown to struggle on out-of-distribution compositional generalization. Compositional data augmentation via example recombination has transferred some prior knowledge about compositionality to such black-box neural models for several semantic parsing tasks, but this often required task-specific engineering or provided limited gains.

We present a more powerful data recombination method using a model called Compositional Structure Learner (CSL). CSL is a generative model with a quasi-synchronous context-free grammar backbone, which we induce from the training data. We sample recombined examples from CSL and add them to the fine-tuning data of a pre-trained sequence-to-sequence model (T5). This procedure effectively transfers most of CSL’s compositional bias to T5 for diagnostic tasks, and results in a model even stronger than a T5-CSL ensemble on two real-world compositional generalization tasks. This results in new state-of-the-art performance for these challenging semantic parsing tasks requiring generalization to both natural language variation and novel compositions of elements.

1 Introduction

Compositional generalization refers to the ability to generalize to novel combinations of previously observed atoms.1 For example, we may ask a model to interpret the instruction “jump twice”, when the atoms “jump” and “twice” were each observed separately during training but never in combination with each other (Lake and Baroni, 2018).

Improving compositional generalization is seen as important for approaching human-like language understanding (Lake et al., 2017; Battaglia et al., 2018) and is practically significant for real-world applications, where models deployed in the wild often need to interpret new combinations of elements not well-covered by expensive and potentially skewed annotated training data (Herzig and Berant, 2019; Yin et al., 2021).

Figure 1: An overview of our method for compositional data augmentation with CSL, a generative model with a QCFG backbone, which is automatically induced from the training data. We show a notional set of original and synthetic examples mapping utterances to programs.
In contrast, specialized architectures with discrete latent structure (Chen et al., 2020; Liu et al., 2020; Nye et al., 2020; Herzig and Berant, 2021; Shaw et al., 2021) have made strides in compositional generalization, but without task-specific engineering or ensembling, the gains have been limited to synthetic diagnostic semantic parsing tasks. Although following SCAN (Lake and Baroni, 2018) a number of increasingly complex and realistic synthetic tasks have been created such as CFQ (Keysers et al., 2020) and COGS (Kim and Linzen, 2020), and there are several approaches that achieve good performance on these tasks, the out-of-distribution generalization ability of state-of-the-art models on real-world, non-synthetic tasks is still far from sufficient (Shaw et al., 2021; Yin et al., 2021).

Given their different strengths and weaknesses, it is compelling to combine the compositional bias of such specialized models with the greater flexibility and ability to handle natural language variation that characterizes generic pre-trained neural sequence-to-sequence models. One method for this is data augmentation. For example, Jia and Liang (2016) generate new training examples using example recombination via induced high-precision synchronous grammars, resulting in improvements on in-distribution and compositional splits for natural data. Another example is GECA (Andreas, 2020), a more general data augmentation approach that does not require task-specific assumptions. GECA achieved further gains on a larger variety of tasks, but provided limited improvements on some compositional generalization challenges.

We present a compositional data augmentation approach that generalizes these earlier approaches. Training examples are recombined using the Compositional Structure Learner (CSL) model, which is a generative model over latent derivations with a (quasi-)synchronous context-free grammar (QCFG) backbone, automatically induced from the training data. CSL is more generally applicable than the method of Jia and Liang (2016), employing a generic grammar search algorithm to explore a larger, higher-coverage space of possible grammars. Unlike GECA, CSL can re-combine examples recursively and also defines a probabilistic sampling distribution over input-output pairs.

The overall approach is illustrated in Figure 1. Given a training set of \( \langle x, y \rangle \) pairs—for example, pairs of natural language utterances and semantic parses—CSL is automatically induced by first finding a QCFG through a search method based on an extension of the induction procedure of Shaw et al. (2021). We then estimate a probabilistic generative model that assigns probabilities to latent derivations of \( \langle x, y \rangle \) pairs according to the grammar. Finally, CSL is used to sample additional synthetic examples for training, and the union of original and generated datasets is used to fine-tune the T5 sequence-to-sequence model (Raffel et al., 2020).

Like the NQG model of Shaw et al. (2021), CSL can, on its own, address a variety of compositional generalization diagnostic tasks on synthetic datasets and achieves high precision (but limited recall) on non-synthetic compositional generalization challenges, leading to overall gains when ensembled with T5. Crucially, augmenting the training data for T5 with samples from CSL transfers most of CSL’s compositional bias to T5 for diagnostic tasks and outperforms a T5+CSL ensemble on non-synthetic compositional generalization tasks defined by compositional splits of GeoQuery (Zelle and Mooney, 1996) and SMCalFlow (Andreas et al., 2020; Yin et al., 2021), resulting in new state-of-the-art performance on these splits.

2 Background and Problem Formulation

Evaluations of compositional generalization are often described as assessing generalization to novel “out-of-distribution” combinations of the “atoms” in the training set. We now take steps toward laying out a more precise problem formulation appropriate for the semantic parsing and instruction following tasks we study.

Problem Setting Consider a training dataset \( \mathcal{D} \) consisting of input-output pairs \( \langle x, y \rangle \in \mathcal{X} \times \mathcal{Y} \), where \( \mathcal{X} \) is the set of valid inputs and \( \mathcal{Y} \) is the set of valid outputs.

We assume that \( \langle x, y \rangle \in \mathcal{D} \) are sampled from a source distribution \( p_s(x, y) \). Our model will be evaluated on inputs from a target distribution \( p_t(x, y) \). For a given input \( x \), the model should predict the most likely \( y \) according to \( p_t(y|x) \).

We make an assumption that the conditional distribution of \( y \) given \( x \) is unchanged between source and target distributions; i.e., \( p_s(y|x) = p_t(y|x) \), which is also a standard assumption for domain adaptation evaluations under covariate shift. However, any or all of the following may be true:
will assume that both the training and evaluation
what remains the 
\( \lambda(x_1, y_1). \lambda(x_2, y_2). \langle x_1 \) and \( x_2, y_1, y_2 \) \\
\( \lambda(x_1, y_1), \langle x_1 \text{ twice } , x_1 \rangle \) \\
( jump , JUMP ) \\
( walk , WALK ) 

Figure 2: An example derivation that derives the string pair (jump twice and walk , JUMP JUMP WALK).

\( p_s(x, y) \neq p_t(x, y) \), \( p_s(x) \neq p_t(x) \), \( p_s(y) \neq p_t(y) \), and \( p_s(x|y) \neq p_t(x|y) \).

What differentiates our setting from other forms of distribution shift is the added assumption that the source and target distributions share common atoms. As a guiding principle, it would be helpful to have a more precise description of atoms and how they are combined, and a clearer theory about what remains the same between the training and evaluation distributions.

We propose initial steps towards formally defining this problem. We will assume that these “atoms” are functions, which can be recombined via function application to form new pairs of inputs and outputs. Next, we define a general class of models termed derivational generative models, as a possible formulation to make this notion precise. We will assume that both the training and evaluation distributions can be modeled by derivational generative models that share a common set of underlying functions.

**Derivational Generative Models** A derivational generative model defines a distribution \( p(x, y) \) over input-output pairs. The model contains a set of functions, \( \mathcal{G} \), and a distribution over derivations. A derivation \( d \) can be viewed as a tree of functions from \( \mathcal{G} \) which derives some element \( \langle z \rangle = \langle x, y \rangle \in \mathcal{X} \times \mathcal{Y} \) determined by recursively applying the functions in \( d \). An example derivation is shown in Figure 2.

Given \( \mathcal{X}, \mathcal{Y}, \) and \( \mathcal{G} \), we can generate a set \( \mathcal{Z}^G \) of possible derivations. We define some shorthands for important subsets of \( \mathcal{Z}^G \) for given \( x \) and \( y \):

\[ \mathcal{Z}^G_{(x,y)} = \{ z \in \mathcal{Z}^G | \langle z \rangle = \langle x, y \rangle \} \]

\[ \mathcal{Z}^G_{(x,y)} = \{ z \in \mathcal{Z}^G | \exists y' \in \mathcal{Y}, \langle z \rangle = \langle x, y' \rangle \} \]

A derivational generative model also consists of some probability distribution \( p_0(z) \) over the set of derivations \( \mathcal{Z}^0 \), which we assume to be parameterized by \( \theta \). We define \( p_{\mathcal{G}, \theta}(x, y) \) in terms of \( p_0(z) \) as:

\[ p_{\mathcal{G}, \theta}(x, y) = \sum_{z \in \mathcal{Z}^g_{(x,y)}} p_0(z), \]

and therefore:

\[ p_{\mathcal{G}, \theta}(y|x) = \frac{\sum_{z \in \mathcal{Z}^g_{(x,y)}} p_0(z)}{\sum_{z \in \mathcal{Z}^g_{(x,y)}} p_0(z)}, \]

for \( p_{\mathcal{G}, \theta}(x) > 0 \).

**Discussion** In general, we are interested in a set of functions that captures some knowledge of how the parts of inputs correspond to parts of outputs. If we can recover some approximation of the underlying set of functions, \( \mathcal{G} \), given \( \mathcal{D} \), then we could sample derivations consisting of new combinations of functions that are not observed in \( \mathcal{D} \). This could potentially help us improve performance on the target distribution, since we assume that the set of functions is unchanged between the source and target distributions, and that what is varying is the distribution over derivations.

However, even assuming \( \mathcal{G} \) can be exactly recovered given \( \mathcal{D} \) is not sufficient to ensure that we can correctly predict the most likely \( y \) given \( x \) according to the true \( p(y|x) \) (shared between source and target distributions) for \( x \sim p_t(x) \). We must also assume that there exists a parameterization of \( p_0(z) \) such that when we estimate \( \theta' \) given \( \mathcal{D} \), \( p_{\mathcal{G}, \theta'}(y|x) \) sufficiently approximates the true \( p(y|x) \) for \( x \sim p_t(x) \). We hypothesize that conditional independence assumptions with respect to how \( p_0(z) \) decomposes across the function applications in \( z \) can be helpful for this purpose. In particular, such assumptions can enable “reusing” conditional probability factors across the exponential space of derivations, potentially improving transfer to the target distribution.

With this intuition in mind, in §3 we propose a specific class of functions for \( \mathcal{G} \) based on (quasi-)synchronous context-free grammars, as well as a parameterization of \( p_0(z) \) with strong conditional independence assumptions.

\( ^3 \)One special case is where \( |\mathcal{Z}^g_{(x,y)}| = 1 \) for all \( x \). In this case, every \( x \) has exactly one unique derivation and \( p_{\mathcal{G}, \theta}(y|x) \) is deterministic given \( \mathcal{G} \) and does not depend on \( \theta \), and therefore recovering \( \mathcal{G} \) is sufficient.
As shown in Figure 1, our method consists of two stages. First, we induce our generative model, CSL, from training data (§3.1). Second, we sample synthetic examples from the generative model and use them to augment the training data for a sequence-to-sequence model (§3.2).

3 Proposed Method

As shown in Figure 1, our method consists of two stages. First, we induce our generative model, CSL, from training data (§3.1). Second, we sample synthetic examples from the generative model and use them to augment the training data for a sequence-to-sequence model (§3.2).

3.1 Compositional Structure Learner (CSL)

CSL can be viewed as a derivational generative model, as defined in §2, where the set \( G \) of recursive functions is defined by a (quasi-)synchronous context free grammar (QCFG).\(^4\) We first describe the grammar formalism and the parameterization of our probabilistic model. Then we describe our two-stage learning procedure for inducing a grammar and learning the model parameters.

3.1.1 Grammar Formalism

Synchronous context-free grammars (SCFGs) have been used to model the hierarchical mapping between pairs of strings in areas such as compiler theory (Aho and Ullman, 1972) and multiple natural language tasks, e.g., machine translation (Chiang, 2007) and semantic parsing (Wong and Mooney, 2006; Andreas et al., 2013). The correspondence between rules over input and output strings in SCFGs is akin to a homomorphism between syntactic and semantic structures, commonly posited by formal theories of compositional semantics (Montague, 1970; Janssen and Partee, 1997).

SCFGs can be viewed as an extension of context-free grammars (CFGs) that synchronously generate strings in what we will refer to as an input and output language. We write SCFG rules as \( NT \rightarrow \langle \alpha, \beta \rangle \), where \( NT \) is a non-terminal symbol, and \( \alpha \) and \( \beta \) are strings of non-terminal and terminal symbols. We restrict our grammars to have only a single nonterminal symbol, \( NT \).

An SCFG rule can be viewed as two CFG rules, \( NT \rightarrow \alpha \) and \( NT \rightarrow \beta \), with a pairing between the occurrences of non-terminal symbols in \( \alpha \) and \( \beta \). This pairing is indicated by assigning each non-terminal in \( \alpha \) and \( \beta \) an index \( i \in \mathbb{N} \). Non-terminals sharing the same index are called linked. Following convention, we denote the index for a non-terminal using a boxed subscript, e.g. \( NT[i] \). An example derivation is shown in Figure 3.

Our grammars can be quasi-synchronous (Smith and Eisner, 2006) because we allow a one-to-many alignment between non-terminals, i.e. a non-terminal in \( \alpha \) can share an index with more than one non-terminal in \( \beta \).\(^5\)

Unlike the formalism of Shaw et al. (2021), which limited \( \alpha \) to contain \( \leq 2 \) non-terminals, in the current work the maximal number of non-terminals in \( \alpha \) is a configurable parameter; we find that 4 is a computationally tractable choice for the datasets we study.

3.1.2 Probabilistic Model

We factorize the probability of a derivation in terms of conditional probabilities of sequentially expanding a rule from its parent. Formally, let \( r \) denote a rule expanded from its parent rule \( r_p \)'s \( NT[i] \) non-terminal (or a special symbol at the root of the derivation tree).\(^6\) We assume conditional independence and factorize the probability of \( z \) as

\[
p_\theta(z) = \prod_{r,r_p,i \in z} p_\theta(r | r_p, i)
\]

This non-terminal annotation with context from the tree is akin to parent annotation or other structure conditioning for probabilistic context-free grammars (Johnson, 1998; Klein and Manning, 2003).

Using independent parameters for each combination of a rule, its parent, and non-terminal index may lead to overfitting to the training set, limiting our ability to generalize to new combinations of rule applications that are needed for compositional generalization. We therefore factor this distribution using a soft clustering into a set of latent states \( S \).

\(^4\) QCFG rules can be interpreted as functions which are essentially limited to string concatenation. For notational convenience, we will therefore treat \( G \) as a set of QCFG rules in §3.

\(^5\) This is important for datasets such as SCAN, as it allows rules such as \( NT \rightarrow \langle NT[i] \text{ twice}, NT[i] | NT[i] \rangle \) which enable repetition.

\(^6\) Using the example derivation from Figure 3, for the rule application \( r = NT \rightarrow \langle \text{walk, WALK} \rangle \), we have \( r_p = NT \rightarrow \langle NT[i], \text{NT}[\text{NT}[i]] \rangle \) and the expansion probability for that rule application is \( p(r | r_p, 2) \).
representing parent rule application contexts:

\[ p_\theta(r | p, i) = \sum_{s \in S} p_\theta(r | s)p_\theta(s | p, i) \]  \hspace{1cm} (4)

where

\[ p_\theta(s | p, i) = \frac{e^{\theta_{s,p,i}}}{\sum_{s' \in S} e^{\theta_{s',p,i}}} \]  \hspace{1cm} (5)

\[ p_\theta(r | s) = \frac{e^{\theta_{r,s}}}{\sum_{r' \in S} e^{\theta_{r',s}}} \]  \hspace{1cm} (6)

where the \( \theta \)s are scalar parameters.

The number of context states \(|S|\) is a hyperparameter, which we analyze in §5. We also optionally consider a task-specific target CFG, \( T_y \), which defines valid output constructions, and can be used to restrict the derivations considered by CSL to those that generate a pair \((x, y)\) where \( y \in T_y \).\(^{7}\)

### 3.1.3 Learning Procedure

A principled method to estimate \( \mathcal{G} \) and \( \theta \) given \( \mathcal{D} \) would be to find the MAP estimate based on some prior, \( p(\mathcal{G}, \theta) \), that encourages compositionality:

\[ \arg \max_{\mathcal{G}, \theta} p(\mathcal{G}, \theta) \times \prod_{(x, y) \in \mathcal{D}} p_{\mathcal{G}, \theta}(x, y) \]  \hspace{1cm} (7)

However, since optimizing \( \mathcal{G} \) and \( \theta \) jointly is computationally challenging, we adopt a two-stage process similar to that of Shaw et al. (2021).

First, we learn an unweighted grammar using a surrogate objective for the likelihood of the data and a compression-based compositional prior that encourages smaller grammars that reuse rules in multiple contexts, inspired by the Minimum Description Length (MDL) principle (Rissanen, 1978; Grunwald, 2004). We describe the induction objective and algorithm in §3.1.4.

Second, given a grammar \( \mathcal{G} \), we optimize the parameters \( \theta \) by maximizing the log-likelihood of \( p_{\mathcal{G}, \theta}(x, y) \), as defined by Eq. 1, using the Adam optimizer (Kingma and Ba, 2015). To optimize \( \theta \) efficiently, we use a variant of the CKY algorithm (Cocke, 1969; Kasami, 1965; Younger, 1967) to determine the set of derivations, represented as a parse forest, and use dynamic programming to efficiently sum over this set.

\(^{7}\)The outputs for several of the tasks we study consist of executable programs or logical terms, for which we can assume the availability of a CFG for parsing. Details can be found in Appendix A.

\[ \langle \text{NT}_{[1]}, \text{NT}_{[1]} \rangle \xrightarrow{r_a} \langle \alpha_b, \beta_b \rangle \]

\[ \langle \text{NT}_{[1]} \rangle \xrightarrow{r_b} \langle \alpha_a, \beta_a \rangle \]

\[ \langle \text{NT}_{[1]} \rangle \xrightarrow{r_c} \langle \text{jump, JUMP} \rangle \]

\[ \langle \text{jump and NT}_{[1]}, \text{JUMP NT}_{[1]} \rangle \]

Figure 4: The arrows in the diagram denote expansion of a nonterminal with a rule. When the above ternary relation holds between \( r_a \), \( r_b \), and \( r_c \), such as in the provided example, we will write \( r_a \circ r_b \Rightarrow r_c \). The key sub-routine of our grammar induction algorithm, UNIFY\((r_1, r_2)\), returns the set of rules \( \{r_3 | r_2 \circ r_3 \Rightarrow r_1 \} \).

### 3.1.4 Grammar Induction Algorithm

Our method for inducing a QCFG is based on that of Shaw et al. (2021), but with several modifications, which improve the computational scalability of the algorithm as well as the precision and coverage of the induced grammar. We analyze the relative performance of the two algorithms in §5.

**Objective** The main idea of the grammar induction objective, \( L_D(\mathcal{G}) \), is to balance the size of the grammar with its ability to fit the training data:

\[ L_D(\mathcal{G}) = \sum_{\text{NT} \rightarrow (\alpha, \beta) \in \mathcal{G}} |\alpha| + |\beta| - c_D(\alpha, \beta), \]  \hspace{1cm} (8)

where \(|\cdot|\) is a weighted count of terminal and nonterminal tokens (the relative cost of a terminal vs. nonterminal token is a hyperparameter) and:

\[ c_D(\alpha, \beta) = k_\alpha \ln \hat{p}_D(\alpha | \beta) + k_\beta \ln \hat{p}_D(\beta | \alpha) \]  \hspace{1cm} (9)

where \( k_\alpha \) and \( k_\beta \) are hyperparameters and \( \hat{p}_D(\alpha | \beta) \) is equal to the fraction of examples \((x, y) \in \mathcal{D}_{\text{train}}\) where \( \alpha \) “occurs in” \( x \) out of the examples where \( \beta \) “occurs in” \( y \), and vice versa for \( \hat{p}_D(\beta | \alpha) \).\(^{8}\)

The correlation between \( \alpha \) and \( \beta \) as measured by the \( \hat{p} \) terms provides a measure related to how well the rule fits the training data. We use sampling to optimize the computation of \( \hat{p}_D \) for larger datasets.

\(^{8}\)By \( \alpha \) “occurs in” \( x \), we mean that there exists some substitution for any non-terminals in \( \alpha \) such that it is a substring or equal to \( x \).
While the primary goal of CSL is to be used to sampling new examples for data augmentation (discussed next), we can also use CSL as a parsing model, by using a variant of the CKY algorithm to find the highest scoring derivation \( z \) that improves Eq. 3 for a given input \( x \). We then output the corresponding \( y \) if it can be derived by the given output CFG, or if no output CFG is provided.

3.2 Data Augmentation

We synthesize a configurable number of examples by sampling from the learned generative model, CSL. For all experiments, we sample 100,000 synthetic examples. To generate a synthetic example \( (x, y) \), we use forward sampling: we start from the single \( NT \) symbol and sample recursively to expand each nonterminal symbol with a rule, based on \( \theta_\delta(r|Y_p,i) \) defined by Eq. 4. If a CFG \( T_y \) defining valid outputs is provided for the given task then we ensure the sampled \( y \) can be generated by \( T_y \).

Given that sequence-to-sequence models perform especially poorly on length extrapolation (Newman et al., 2020), we optionally bias our sampling to favor deeper derivations. We achieve this by adding a bias \( \delta > 0 \) to each \( \theta_\delta(r,i) \) where the rule \( r \) contains greater than a configurable number of nonterminals.

We fine-tune T5 on the union of the original training data and the synthesized data. Following Jia and Liang (2016), we ensure an approximately equal number of original and synthesized examples are used for training. We achieve this by replicating original or synthetic examples as needed.

4 Experiments

Our main proposed method, T5+CSL-Aug., uses CSL to generate examples for augmenting the training data of T5.\(^{11}\) To comparatively evaluate and analyze this method, we perform two sets of experiments presented in this and the following section.

Experiments and Analysis Overview

First, in the current \S 4 we report on experiments evaluating T5+CSL-Aug., in comparison to T5+GECA, a method augmenting training data with GECA which is prior state of the art for data augmentation (Andreas, 2020), and other representative methods on several synthetic and real-world semantic parsing datasets. Here we aim to understand the potential of the method to advance the state of the art in compositional generalization evaluations.

Then in \S 5, we delve deeper to analyze the contribution of CSL as a scoring model in a T5+CSL ensemble versus its contribution as a data augmentation model for T5, studying CSL’s impact on inputs within or outside the language defined by
CSL’s grammar. There we also evaluate the novel aspects of CSL’s grammar search and probabilistic model parametrization compared to its close predecessor NQG, and their impact on speed, coverage, and parsing accuracy. Finally, we measure the impact of context sensitivity in CSL’s generative model on the ability of the model to extrapolate to valid out-of-distribution compositions.

4.1 Synthetic Evaluations

We first evaluate our approach on synthetic benchmarks designed for controlled assessments of compositional generalization. In these benchmarks, subsidiary challenges such as language variation are minimized, as the training and evaluation datasets are generated by a shared set of rules.

4.1.1 Datasets

SCAN The SCAN dataset contains navigation commands paired with action sequences. We consider three compositional data splits from Lake and Baroni (2018): the jump and turn left splits (where a new primitive is used in novel combinations), and the length split (where test sequences are longer than training ones). We also consider the MCD split from Keysers et al. (2020) created by making the distributions of compositional structures in training and test data as divergent as possible.

COGS The COGS dataset (Kim and Linzen, 2020) contains sentences paired with logical forms. We use the generalization test set, which tests generalization to novel linguistic structures. As SCFG cannot handle logical variables (Wong and Mooney, 2007), we convert the outputs into equivalent variable-free forms, detailed in Appendix A.

4.1.2 Results

The results are shown in Table 1. We also compare with previously reported results for methods other than data augmentation. For SCAN, NQG-T5 Shaw et al. (2021) is one of several specialized models that achieves 100% accuracy across multiple splits (Chen et al., 2020; Liu et al., 2020; Nye et al., 2020; Herzig and Berant, 2021). We also report new results for NQG-T5 on COGS, and show results from LeAR (Liu et al., 2021), the previously reported state-of-the-art on COGS.

For these synthetic datasets, the induced grammars have high coverage, making the CSL model highly effective for data augmentation. When we use CSL to generate additional training data for T5, the performance of T5 improves to nearly solving SCAN and sets new state-of-the-art results on COGS.

4.2 Non-Synthetic Evaluations

Non-synthetic, real-world evaluations of compositional generalization introduces the additional challenge of handling natural language variation. For example, some words in the test data might never appear during training, making it challenging to induce a grammar with high coverage on the test data.

4.2.1 Datasets

GeoQuery The GeoQuery dataset (Zelle and Mooney, 1996; Tang and Mooney, 2001) contains human-authored questions paired with meaning representations. We report results on the standard data split as well as three compositional splits based on those introduced in Shaw et al. (2021): the template split (where abstract output templates in training and test data are disjoint (Finegan-Dollak et al., 2018)), the TMCD split (an extension of MCD for non-synthetic data), and the length split. We average results across 3 TMCD and Template splits generated with different random seeds to reduce variance due to small dataset sizes. Details are in Appendix A.

SMCalFlow-CS Yin et al. (2021) proposed a compositional skills split of the SMCalFlow dataset (Andreas et al., 2020) that contains dialog turns paired with LISP programs. Each training example is a single-turn sentence from one of two domains related to creating calendar events or querying an org chart. The examples are filtered such that they do not require conversational context. The single-domain (S) test set has examples from a single domain, while the cross-domain (C) test set has sentences that require knowledge from both domains (e.g. “create an event with my manager”). Since cross-domain language patterns might not
be deducible from single-domain training examples, the benchmark has a small amount (8, 16, or 32) of labeled examples from the test distribution added to the training data. Further details are in Appendix A.

4.2.2 Results

As shown in Table 2, using CSL to generate additional training data for T5 (T5+CSL-Aug.) improves the accuracy of T5 and achieves state-of-the-art on most splits. Using CSL for data augmentation outperforms using GECA for data augmentation. (Note that GECA was not computationally feasible to run on SMCalFlow.) T5+CSL-Aug. also outperforms the previous state-of-the-art, the ensemble approach NQG-T5 of (Shaw et al., 2021).

For SMCalFlow-CS, we also show previously reported results by Yin et al. (2021). These include a seq2seq model with a BERT (Devlin et al., 2019) encoder and the coarse2fine (C2F) model of Dong and Lapata (2018) as baselines, as well as each method combined with the span-supervised (SS) attention method of Yin et al. (2021).

5 Analysis and Discussion

5.1 Performance Breakdown

In Table 3, we analyze the relative performance of T5, CSL, and combinations of T5 and CSL using ensembling and data augmentation across inputs that can be generated by CSL and ones that cannot.

CSL can produce outputs only when the input is covered by the grammar, while T5 trained on original training data performs well on in-distribution inputs but struggles for out-of-distribution ones. An ensemble model works well on the inputs covered by either CSL or T5, but cannot generalize beyond such inputs.

Our data augmentation procedure is akin to knowledge distillation. In particular, we distill the ability to compositionally generalize to out-of-distribution inputs from the teacher model CSL to the student model T5. With data augmentation, T5+CSL-Aug. makes accuracy gains on inputs both within and outside the scope of the grammar. This means that the model is, to some degree, generalizing from the distillation data \( x \in \mathcal{X}_{CSL} \) to \( x \notin \mathcal{X}_{CSL} \).

5.2 Varying Context Sensitivity

Varying the number of context states \( |S| \) can vary the degree of context sensitivity in the model. This can be important because we want our model to be able to accurately model \( p(y|x) \), which we assume is shared between the source and target distributions, but we also want to sample new inputs \( x \) that may have low probability under the source distribution due to the novel compositions they contain.

As a step towards understanding the trade-offs related to context sensitivity, we compute the accuracy of CSL, as well as the log likelihood of the training and dev sets of the SCAN MCD1 split in Table 4, for CSL models with different number of context clusters \( |S| \).

A constraint on the number of context states \( |S| \) is in some ways similar to a constraint on the number of nonterminal symbols in a conventional SCFG. Notably, for SCAN, writing a SCFG that unambiguously maps inputs to outputs requires 2 unique nonterminal symbols, and we observe that, similarly, \( |S| \geq 2 \) is required to reach 100% accuracy on the dev set. We also observe that while the models with larger \( |S| \) fit the training set better, the log likelihood of the dev set is highest at \( |S| = 4 \), indicating that the optimal place on the tradeoff curve is not at the extremes.

It is also worth noting that, regardless of the
5.3 Comparing CSL and NQG

CSL and NQG of Shaw et al. (2021) vary across several dimensions, as the two systems use different grammar induction algorithms and different model parameterizations. Here we compare the two approaches across both dimensions independently.

Grammar Induction As discussed in §3.1.4, the largest set of changes to the CSL algorithm from that of NQG were to improve the scalability of the induction algorithm, as both algorithms scale super-linearly in both dataset size and the length of input and output strings. The runtime of grammar induction on the GeoQuery standard split on a standard workstation CPU is around 15 minutes for NQG, and \(< 1\) minute for CSL. More importantly, we did not find it feasible to run NQG for SMCalFlow-CS, while CSL enables grammar induction to be completed in a couple of days. Further optimizations and parallelization of the algorithm could potentially enable further scaling useful for large datasets, such as CFQ discussed in §5.5.

CSL also supports QCFG rules with \(\geq 2\) nonterminals while NQG does not. We found allowing up to 4 nonterminals can improve the coverage of induced grammars for COGS and SMCalFlow-CS, with some example rules shown in Table 8 in Appendix A. Notably, for COGS, the induction algorithm of NQG induced a grammar that can only...
derive 64.9% of the test set.

Model Parameterization We compare the performance of CSL’s simple generative model with the span-based model of NQG which uses a BERT-Base encoder in Table 5 using the same grammar induced using the CSL algorithm. Overall, the models perform comparable, despite CSL having far fewer parameters, not leveraging pre-trained neural networks, and being a generative model to support sampling (as opposed to a discriminative model only in the case of NQG). However, NQG appears to be better at resolving ambiguity related to the bracketing of inputs for the longer inputs in the GeoQuery length split. This is perhaps unsurprising since span-based neural models perform very well at constituency parsing (Stern et al., 2017). Incorporating pre-trained neural components into a model such as CSL would be a promising future direction.

For SMCalFlow-CS, given a grammar induced by CSL, we found that it can be computationally infeasible to train a discriminative NQG model, due to the need to compute the partition function which sums over $Z^{\mathcal{G}}_{x,s}$. As a generative model, CSL avoids the need to compute a partition function during training.

5.4 Comparison with GECA

From Tables 1 and 2 we see that augmenting the training data using CSL outperforms GECA across tasks on both synthetic and non-synthetic evaluations. GECA relies on the simple assumption that fragments are interchangeable if they appear in the same context. It is restricted by a pre-defined window size for fragments, and does not support recursion. Figure 5 compares differences in the sets of derivable synthetic examples for a notional set of training examples. CSL can derive a significantly larger set of synthetic examples than GECA, and also assigns a probability to each derivable example.

$\mathcal{G}^T \equiv \{(\text{jumps, JUMP}), (\text{walk, WALK})\}$

Figure 5: We show the set of derivable synthetic examples for GECA (with window size = 2) and CSL, given an illustrative example of training examples. CSL can derive a significantly larger set of synthetic examples than GECA, and also assigns a probability to each derivable example.

rule $\text{NT} \rightarrow (\text{NT}[1] \text{ and NT}[2])$, which can be applied recursively.

Additionally, we found GECA was not computationally feasible to run for COGS and SMCalFlow, as the algorithm’s iteration over templates and fragments becomes prohibitive for larger-scale datasets.

5.5 Limitations of QCFGs

The mapping from inputs to outputs in SCAN, COGS, and GeoQuery are all well supported by QCFGs. However, grammars were used to generate the data for SCAN and COGS, so this is perhaps not surprising. While GeoQuery inputs were written by humans, the distribution of queries in the dataset is influenced by the capabilities of the underlying execution engine based on logic programming; the dataset has a large number of nested noun phrases in inputs that map directly to nested FunQL clauses in outputs.

SMCalFlow The induced grammars have relatively low coverage on SMCalFlow, as shown by $\mathcal{G}^{\mathcal{X}_{\text{CSL}}}$ in Table 3, although they are still sufficient to improve the performance of T5. One reason for the low coverage is that inputs in SMCalFlow often reference specific names, locations, and meeting subjects, such as “setup up a sales meeting with Sam and his manager” where “sales meeting” and “Sam” must be copied to the output program as string literals. Sequence-to-sequence models with copy mechanisms or shared input-output vocabularies can handle such copying, but the QCFGs induced by our method do not support generaliza-
tion to such novel tokens. Extending the method to support such string copying could significantly improve coverage.

Another reason for the low coverage is that the mismatch between the nesting of prepositional phrases in the input (e.g., “at NT” and “with NT”) and the corresponding clauses in the output program tree makes it difficult to induce QCFG rules that enable recombination of different prepositional phrases in different contexts.

**CFQ** We also evaluated the feasibility of our approach to improve T5 performance on CFQ (Keyser et al., 2020), a popular synthetic dataset for evaluating compositional generalization. We found it was challenging to induce QCFGs with reasonable coverage for CFQ. First, the SPARQL queries in CFQ contain variables, which are not well supported by QCFGs (Wong and Mooney, 2007). Additionally, the mapping from queries to SPARQL in CFQ requires notions of commutativity (both “M0 edited and directed M1” and “M0 directed and edited M1” will be mapped to “M0 ns:film.director.film M1 . M0 ns:film.editor.film M1””) and distributivity (edited in “edited M1 and M2” will appear twice in “?x0 ns:film.editor.film M1 . ?x0 ns:film.editor.film M2”) that are also not well supported by QCFGs. Such limitations can potentially be partially overcome by designing intermediate representations for CFQ (Furrer et al., 2020; Herzig et al., 2021), but a complete solution likely requires an extension to the class of allowable rules in $G$ beyond those a QCFG formalism supports, such as better support for variables (Wong and Mooney, 2007) and the ability to apply rewriting rules to generated output strings.

6 Related Work

**Grammar Induction** Prior to the trend towards sequence-to-sequence models, significant prior work in semantic parsing explored inducing SCFG (Wong and Mooney, 2006, 2007; Andreas et al., 2013) and CCG (Zettlemoyer and Collins, 2005, 2007; Kwiatkowski et al., 2010; Kwiatkowski et al., 2013; Artzi et al., 2014) grammars. SCFGs have also been applied to machine translation (Chiang, 2007; Blunsom et al., 2008; Saers et al., 2013). Compression-based objectives similar to ours have also been applied to CFG induction (Grünwald, 1995). Recently, the method of Kim (2021) learns neural parameterized QCFG grammars, which can avoid the pitfalls in coverage of lexicalized grammars such as the ones we learn; however the approach can be computationally demanding for longer input-output pairs.

**Data Augmentation** Data augmentation has been widely used for semantic parsing and related tasks (Jia and Liang, 2016; Andreas, 2020; Akyürek et al., 2021; Wang et al., 2021b; Zhong et al., 2020; Oren et al., 2021; Tran and Tan, 2020; Guo et al., 2020, 2021). Jia and Liang (2016) perform data recombination using an induced SCFG but their approach requires domain-specific heuristics. GECA (Andreas, 2020) provides a more general solution, which we analyzed in §5. The data recombination method of Akyürek et al. (2021) is appealing because it can learn recombinations without committing to a specific grammar formalism, although gains were limited relative to symbolic methods. The SeqMix approach of Guo et al. (2020) also learns to recombine training examples; it demonstrates gains for translation tasks but is not as effective as GECA for semantic parsing tasks. Other approaches leverage a forward semantic parser and a backward input generator with some variants (Wang et al., 2021b; Zhong et al., 2020; Tran and Tan, 2020; Guo et al., 2021), but most of these approaches do not explicitly explore the compositional generalization setting. Oren et al. (2021) propose an approach to sample more structurally-diverse data to improve compositional generalization, given a manually specified SCFG.

**Compositional Generalization** Many approaches have been pursued to improve compositional generalization in semantic parsing, beyond the previously discussed data augmentation methods. These include alternative model architectures (Li et al., 2019; Russin et al., 2019; Gordon et al., 2020; Liu et al., 2020; Nye et al., 2020; Chen et al., 2020; Zheng and Lapata, 2020; Oren et al., 2020; Herzig and Berant, 2021; Ruiz et al., 2021; Wang et al., 2021a), different Transformer variations and configurations (Csordás et al., 2021; Ontanón et al., 2021), ensemble models (Shaw et al., 2021), intermediate representations (Herzig et al., 2021), meta-learning (Lake, 2019; Conklin et al., 2021; Zhu et al., 2021), and auxiliary objectives to bias attention in encoder-decoder models (Yin et al., 2021; Jiang and Bansal, 2021). Also, Furrer et al. (2020) compare pre-trained models vs specialized architectures for compositional generalization.
7 Conclusion

We showed that the Compositional Structure Learner (CSL) generative model improves the state of the art on compositional generalization challenges for two real-world semantic parsing datasets when used to augment the task training data for the generic pre-trained T5 model. Data augmentation using CSL was also largely sufficient to distill CSL’s knowledge about structure into T5 for multiple synthetic compositional generalization evaluations. While CSL has limitations (notably, the QCFG formalism is not a good fit for all phenomena in the mapping of natural language to corresponding logical forms), our experiments suggest the strong potential of more powerful probabilistic models over automatically induced latent structures as data generators for black-box pretrained sequence-to-sequence models.

Acknowledgements

We thank Nitish Gupta, Ming-Wei Chang, William Cohen, Waleed Ammar, and the Google Research Language team for helpful comments and discussions.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. The theory of parsing, translation, and compiling, volume 1. Prentice-Hall Englewood Cliffs, NJ.

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2021. Learning to recombine and resample data for compositional generalization. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7556–7566, Online. Association for Computational Linguistics.

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbak, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. Transactions of the Association for Computational Linguistics, 8:556–571.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 47–52, Sofia, Bulgaria. Association for Computational Linguistics.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1273–1283, Doha, Qatar. Association for Computational Linguistics.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Matus Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. ArXiv preprint, abs/1806.01261.

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. A discriminative latent variable model for statistical machine translation. In Proceedings of ACL-08: HLT, pages 200–208, Columbus, Ohio. Association for Computational Linguistics.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, 19(2):263–311.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. 2020. Compositional generalization via neural-symbolic stack machines. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.

David Chiang. 2007. Hierarchical phrase-based translation. Computational Linguistics, 33(2):201–228.

John Cocke. 1969. Programming languages and their compilers: Preliminary notes. New York University.

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 (Volume 1: Long Papers), pages 3322–3335, Online. Association for Computational Linguistics.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages
619–634, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. **BERT: Pre-training of deep bidirectional transformers for language understanding.** In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. **Coarse-to-fine decoding for neural semantic parsing.** In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. **Improving text-to-SQL evaluation methodology.** In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

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

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2020. **Permutation equivariant models for compositional generalization in language.** In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net.

Peter Grünwald. 1995. A minimum description length approach to grammar inference. In International Joint Conference on Artificial Intelligence, pages 203–216. Springer.

Peter Grunwald. 2004. A tutorial introduction to the minimum description length principle. arXiv preprint math/0406077.

Demi Guo, Yoon Kim, and Alexander Rush. 2020. **Sequence-level mixed sample data augmentation.** In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 5547–5552, Online. Association for Computational Linguistics.

Yinuo Guo, Huaiei Zhu, Zeqi Lin, Bei Chen, Jian-Guang Lou, and Dongmei Zhang. 2021. Revisiting iterative back-translation from the perspective of compositional generalization. In AAAI.

Jonathan Herzig and Jonathan Berant. 2019. Don’t paraphrase, detect! rapid and effective data collection for semantic parsing. 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), pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2021. **Span-based semantic parsing for compositional generalization.** 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), pages 908–921, Online. Association for Computational Linguistics.

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, pages 6253–6265, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mark Johnson. 1998. **PCFG models of linguistic tree representations.** Computational Linguistics, 24(4):613–632.

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In Proceedings of the National Conference on Artificial Intelligence, volume 20, page 1062. Menlo Park, CA; Cambridge, MA; London: AAAI Press; MIT Press; 1999.

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. OpenReview.net.
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), pages 9087–9105, Online. Association for Computational Linguistics.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In Advances in Neural Information Processing Systems.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 1223–1233, Cambridge, MA. Association for Computational Linguistics.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1545–1556, Seattle, Washington, USA. Association for Computational Linguistics.

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, pages 9788–9798.

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 of Proceedings of Machine Learning Research, pages 2879–2888. PMLR.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Building machines that learn and think like people. Behavioral and brain sciences, 40.

Yuanpeng Li, Liang Zhao, Jianyu Wang, and Joel Hestness. 2019. 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), pages 4293–4302, Hong Kong, China. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. ArXiv preprint, abs/1309.4408.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning algebraic recombination for compositional generalization. In Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 1129–1144, Online. Association for Computational Linguistics.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.

Richard Montague. 1970. Universal grammar. Theoria, 36(3):373–398.

Benjamin Newman, John Hewitt, Percy Liang, and Christopher D. Manning. 2020. The EOS decision and length extrapolation. In Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 276–291, Online. Association for Computational Linguistics.

Maxwell I. Nye, Armando Solar-Lezama, Josh Tenenbaum, and Brenden M. Lake. 2020. Learning compositional rules via neural program synthesis. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.

Santiago Ontañón, Joshua Ainslie, VACLav Cvicek, and Zachary Fisher. 2021. Making transformers solve compositional tasks. ArXiv preprint, abs/2108.04378.

Inbar Oren, Jonathan Herzig, and Jonathan Berant. 2021. Finding needles in a haystack: Sampling structurally-diverse training sets from synthetic data for compositional generalization. In EMNLP.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. 2020. Improving compositional generalization in semantic parsing. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2482–2495, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21:1–67.
Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14(5):465–471.

Luana Ruiz, Joshua Ainslie, and Santiago Ontañón. 2021. Iterative decoding for compositional generalization in transformers. *ArXiv preprint*, abs/2110.04169.

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

Markus Saers, Karteek Addanki, and Dekai Wu. 2013. Unsupervised transduction grammar induction via minimum description length. In *Proceedings of the Second Workshop on Hybrid Approaches to Translation*, pages 67–73, Sofia, Bulgaria. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. 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)*, pages 922–938, Online. Association for Computational Linguistics.

David Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 23–30, New York City. Association for Computational Linguistics.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pages 466–477. Springer.

Ke Tran and Ming Tan. 2020. Generating synthetic data for task-oriented semantic parsing with hierarchical representations. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages 17–21, Online. Association for Computational Linguistics.

Bailan Wang, Mirella Lapata, and Ivan Titov. 2021a. Structured reordering for modeling latent alignments in sequence transduction. *Advances in Neural Information Processing Systems*, 34.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021b. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanuel 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*, pages 2810–2823, Online. Association for Computational Linguistics.

Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 2*, pages 1050–1055.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666. AUAI Press.

Hao Zheng and Mirella Lapata. 2020. Compositional generalization via semantic tagging. *ArXiv preprint*, abs/2010.11818.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.
Wang Zhu, Peter Shaw, Tal Linzen, and Fei Sha. 2021. Learning to generalize compositionally by transferring across semantic parsing tasks. ArXiv preprint, abs/2111.05013.

Appendix

A Dataset and Preprocessing Details

In this section we detail preprocessing for each dataset. Dataset sizes are reported in Table 6. We show examples of each dataset in Table 7, with examples of the corresponding induced QCFG rules in Table 8.

SCAN We did not perform any preprocessing for SCAN. Grammar induction does not use any seed rules, and we do not assume a CFG defining valid output constructions, as the outputs consist of action sequences, not executable programs or logical forms.

COGS For COGS, as QCFGs do not support logical variables (Wong and Mooney, 2007), we mapped the original logical forms to a variable-free representation, with an example shown in Table 7. The mapping is deterministic and reversible, and is akin to the use of other variable-free logical forms for semantic parsing such as FunQL (Kate et al., 2005) or Lambda-DCS (Liang, 2013). An alternative but potentially more complex solution to handling logical variables in outputs would be to use an extension of SCFGs, such as \(\lambda\)-SCFG (Wong and Mooney, 2007).

We define an output CFG based on the definition of this variable-free representation. To minimize the linguistic priors we did not distinguish the types of primitives (e.g., nouns vs verbs); they all belong to the same CFG category. We use a set of seed rules of the form \(NT \rightarrow \langle x', x \rangle\) where \(x\) is a token found in a training output, and \(x'\) is \(x\) or an inflected form of \(x\) found in a training input (e.g., for \(x = \"sleep\", \) we add \(NT \rightarrow \langle sleep, sleep \rangle\) and \(NT \rightarrow \langle slept, sleep \rangle\)). These \(\langle x', x \rangle\) pairs were identified by running the IBM I alignment model (Brown et al., 1993) on the training data.

GeoQuery We use the same variant of FunQL (Kate et al., 2005) as Shaw et al. (2021), with entities replaced with placeholder values. We generate new length, template, and TMCD splits following the methodology of Shaw et al. (2021), so that we could evaluate our method on dev sets, which the original splits did not include. Specifically, for the length split, we randomly split the test set of the original length split into a dev set of 110 examples and a test set of 330 examples. To reduce variance, we created 3 new template and TMCD splits with different random seeds, targeting 440 training examples, and 440 examples that are then randomly split into a 110 dev set and 330 test set. For the TMCD splits, we changed the atom constraint slightly, based on the error analysis in Shaw et al. (2021) which found that a disproportionate amount of the errors on the TMCD test set were in cases where an “atom” was seen in only a single context during training. To create a fairer evaluation of compositional generalization, we strengthen the atom constraint such that every atom in the test set must be seen at least 2 times in the training set. Additionally, as several function symbols in FunQL can be used with and without arguments, and these usages are semantically quite different, we treat function symbols used with different numbers of arguments as different atoms.

We define an output CFG based on the definition of the FunQL operators and the primitive types in the geobase database. We use a set of seed rules of the form \(NT \rightarrow \langle x, x \rangle\) where \(x\) occurs in both the input and output of a training example. For the length split, we use 1 additional seed rule, \(NT \rightarrow \langle NT_1[1], answer (NT_1) \rangle\), which enables “hallucinating” the “answer” function that wraps all FunQL programs.

| Dataset   | Split       | Train | Dev | Test |
|-----------|-------------|-------|-----|------|
| SCAN      |             |       |     |      |
| Jump      | 14670       | —     | 7706|
| Turn Left | 21890       | —     | 1208|
| Length    | 11990       | —     | 3920|
| MCD1      | 8365        | 1046  | 1045|
| MCD2      | 8365        | 1046  | 1045|
| MCD3      | 8365        | 1046  | 1045|
| COGS      |             |       |     |      |
| Gen       | 95K         | 12K   | 12K |
| GeoQuery  |             |       |     |      |
| Standard  | 600         | —     | 280 |
| Template1 | 438         | 112   | 330 |
| Template2 | 439         | 111   | 330 |
| Template3 | 440         | 110   | 330 |
| TMCD1     | 440         | 110   | 330 |
| TMCD2     | 440         | 110   | 330 |
| TMCD3     | 440         | 110   | 330 |
| Length    | 440         | 110   | 330 |
| SMCalFlow-CS |         |       |     |      |
| 8-shot    | 25412       | 1324  | 1325|
| 16-shot   | 25420       | 1324  | 1325|
| 32-shot   | 25436       | 1324  | 1325|

Table 6: Sizes of all datasets and splits.
| Dataset       | Example                                                                 |
|--------------|-------------------------------------------------------------------------|
| **SCAN**     | \(x:\) walk around right and jump thrice                                   
|              | \(y:\) RTURN WALK RTURN WALK RTURN WALK JUMP JUMP JUMP                   |
| **COGS**     | \(x:\) Camila gave a cake in a storage to Emma.                          
|              | \(y:\) give (agent = Camila, theme = cake (nmod. in = storage), recipient = Emma) |
| **GeoQuery** | \(x:\) what states border states that the m0 runs through                
|              | \(y:\) answer (intersection (state, next_to_2 (intersection (state, traverse_1 (m0))))) |
| **SMCalFlow-CS** | \(x:\) create work meeting with my boss                                  
|              | \(y:\) (Yield: output (CreateCommitEventWrapper:event (CreatePreflightEventWrapper:constraint (Constraint[Event]:attendees (AttendeeListHasRecipient:recipient (FindManager:recipient (toRecipient (CurrentUser)))):subject (?=# (String "work meeting"))))) |

Table 7: Example inputs, \(x\), and outputs, \(y\).

| Dataset       | Induced Rules                                                                 |
|--------------|-------------------------------------------------------------------------------|
| **SCAN**     | \(NT \rightarrow \langle NT[1] \text{ and NT[2]}, NT[1] \text{ NT[2]} \rangle\) |
|              | \(NT \rightarrow \langle NT[1] \text{ thrice, NT[1] NT[2]} \text{ NT[2]} \rangle\) |
|              | \(NT \rightarrow \langle NT[1] \text{ around right, RTURN NT[1]}, \text{ RTURN NT[1]}, \text{ RTURN NT[1]} \rangle\) |
| **COGS**     | \(NT \rightarrow \langle \text{what NT[1] border NT[2]}, \text{ answer (intersection (NT[1], next_to_2 (NT[2])))} \rangle\) |
|              | \(NT \rightarrow \langle \text{intersection (NT[1], NT[2])} \rangle\) |
|              | \(NT \rightarrow \langle \text{that NT[1] runs through, traverse_1 (NT[1])} \rangle\) |
| **GeoQuery** | \(NT \rightarrow \langle \text{boss, FindManager :recipient (NT[1])} \rangle\) |
|              | \(NT \rightarrow \langle \text{NT[1] with NT[2]}, \text{CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event]:attendees (AttendeeListHasRecipient :recipient (NT[2])) :subject (?=# (NT[1]))))} \rangle\) |
|              | \(NT \rightarrow \langle \text{create NT[1]}, \text{ (Yield: output (NT[1]))} \rangle\) |

Table 8: Examples of induced grammar rules for each example in Table 7. Rules with only terminals are omitted for brevity.

**SMCalFlow-CS**  We use the original LISP programs provided with the dataset as the output representation. We extract seed rules for string literals and numbers that are copied from inputs to outputs, such as person names and meeting subjects. Similar to the GeoQuery length split, we add 5 seed rules with a single non-terminal on the input side that enable “hallucinating” various program fragments. We construct an output CFG based on the bracketing of LISP programs and a mapping of argument slots to nonterminals.