# Strengthening Structural Inductive Biases by Pre-training to Perform Syntactic Transformations

**Matthias Lindemann**[1] and **Alexander Koller**[2] and **Ivan Titov**[1,3]
[1] ILCC, University of Edinburgh, [2] LST, Saarland University, [3] ILLC, University of Amsterdam
m.m.lindemann@sms.ed.ac.uk, koller@coli.uni-saarland.de, ititov@inf.ed.ac.uk

## Abstract

Models need appropriate inductive biases to effectively learn from small amounts of data and generalize systematically outside of the training distribution. While Transformers are highly versatile and powerful, they can still benefit from enhanced structural inductive biases for seq2seq tasks, especially those involving syntactic transformations, such as converting active to passive voice or semantic parsing. In this paper, we propose to strengthen the structural inductive bias of a Transformer by intermediate pre-training to perform synthetically generated syntactic transformations of dependency trees given a description of the transformation. Our experiments confirm that this helps with few-shot learning of syntactic tasks such as chunking, and also improves structural generalization for semantic parsing. Our analysis shows that the intermediate pre-training leads to attention heads that keep track of which syntactic transformation needs to be applied to which token, and that the model can leverage these attention heads on downstream tasks.[1]

## 1 Introduction

Inductive biases play a critical role in NLP, particularly in learning from limited data and in systematic generalization beyond the training distribution. While standard seq2seq models excel on in-distribution data, they often lack structural inductive biases and hence perform poorly on structural generalization, i.e. generalization to unseen combinations of known phrases (Keysers et al., 2020), extrapolation to longer inputs (Lake and Baroni, 2018; Hupkes et al., 2020) and deeper recursion (Kim and Linzen, 2020; Li et al., 2023a). While pre-training on large amounts of text improves structural generalization to a certain extent (Furrer et al., 2020), it remains challenging (Yao and Koller, 2022; Li et al., 2023a).

This seems to conflict with observations that pre-training equips models with knowledge about syntax (Tenney et al., 2019; Hewitt and Manning, 2019; Mueller et al., 2022), which should enable structural generalizations. In this paper, we start from the hypothesis that the lack of structural inductive bias is partly due to limited knowledge of how to *use* syntactic information for *structural tasks*.

Traditionally, NLP has heavily relied on syntactic theories and has phrased many tasks as transformations of syntax trees, ranging from conversion of a sentence from active to passive voice (Oliva, 1988) to constructing a semantic representation for a sentence (Montague, 1970). Transformations of syntax trees can address a task in a very generalizable way by using the right abstractions. For example, when constructing the semantic representation of an NP, by the principle of compositionality, the same transformations can be used for NPs whether they serve as direct objects or as indirect objects.

Inspired by this perspective, we propose a new method of strengthening the structural inductive bias of a pre-trained model with an additional intermediate pre-training step to perform syntactic transformations (see Fig. 1). We create a dataset of automatically generated syntactic transformations of English dependency trees. Given a description of the transformation as a prefix and an input sentence, the model is pre-trained to predict the output of the transformation without access to the underlying dependency tree. This pre-training procedure encourages the model to strengthen its representations of syntax and acquire reusable dynamics of syntactic transformations that can be leveraged for downstream tasks. During fine-tuning, gold-standard descriptions of transformations are not available, and we use a prefix of embeddings that are fine-tuned with the rest of the model instead.

[1]We release our code, data and model at https://github.com/namednil/step.

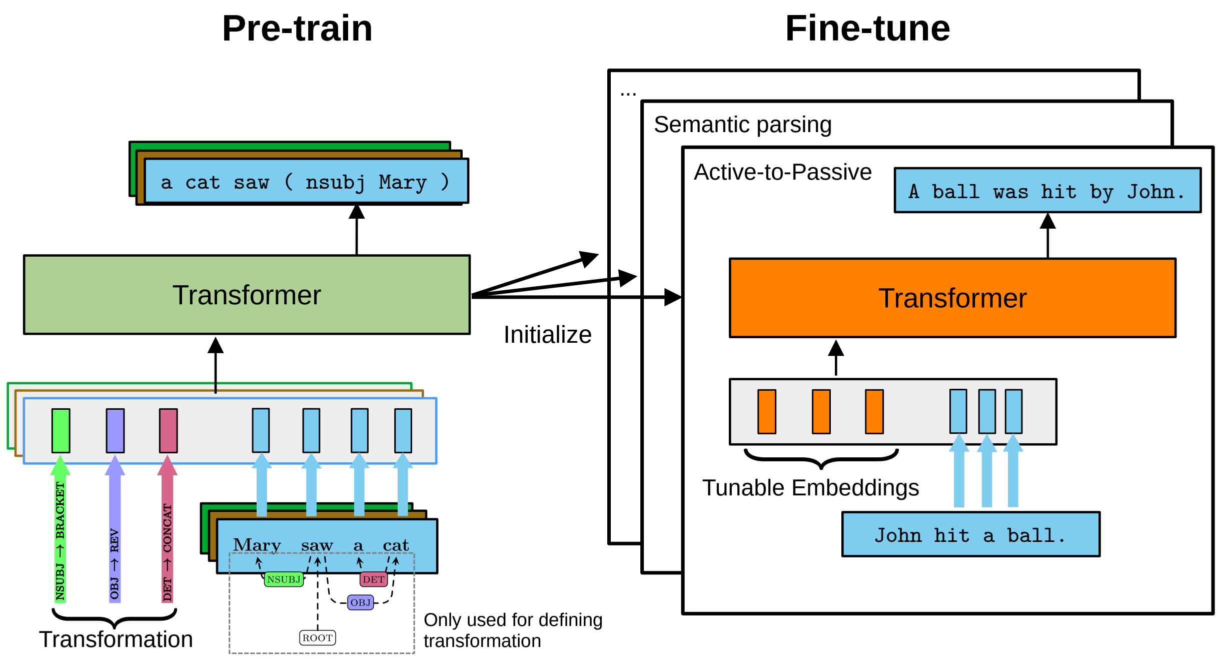

Figure 1: Left: Intermediate pre-training of a Transformer to perform syntactic transformations specified in the prefix; the syntax tree forms the basis of the transformation but is *not* given to the model. Right: fine-tuning the Transformer and the prefix on a downstream task. Tunable parameters are represented in orange.

**Contributions** We demonstrate that our intermediate pre-training strengthens the structural inductive bias of the model, resulting in a better few-shot performance for syntax-dependent seq2seq tasks, such as conversion from active to passive or chunking. Our method also improves structural generalization in the context of semantic parsing.

Analysis of the pre-trained model shows that it uses attention heads to track what transformation needs to be applied to which input token, and that these heads tend to follow syntactic patterns. In addition, we find that fine-tuning re-uses these attention heads, suggesting that the model can leverage the transformations acquired during pre-training.

## 2 Related Work

**Pre-training with synthetic data** Training on synthetic data to shape the inductive bias of Transformers has been explored in several recent works. Papadimitriou and Jurafsky (2023) pre-train on a synthetic language to investigate the impact on language modelling of English. McCoy and Griffiths (2023) pre-train on a distribution of tasks using meta-learning (Finn et al., 2017) and show improvements for low-resource language modelling of child-directed language.

Our work builds conceptually on SIP (Lindemann et al., 2023b), in which a Transformer is pre-trained to simulate the behaviour of Finite State Transducers (FSTs) to introduce a structural inductive bias for FST-like behaviour. That is, given a representation of an automatically generated FST and an input string, a Transformer is pre-trained to predict what the output of the FST is on the given input. During fine-tuning, SIP uses a prefix of tunable embeddings in place of an FST description. While SIP and this work share similar methodology, i.e. pre-training a model with a description of a transformation and fine-tuning the model with a prefix of tunable embeddings, they address different problems: SIP focuses on the sequential inductive bias of FSTs, whereas the present work strengthens the inductive bias for transformations of syntax trees. Another major difference is that SIP's pre-training task is fully deterministic and unambiguous as there is only a single output for any FST and input string. In contrast, here, performing the transformation requires knowledge about the underlying syntax tree, which is not provided to the model. This forces the model to learn the syntax or reuse its existing syntactic knowledge.

**Syntax-infused pre-training** In recent years, several works have explored injecting syntactic knowledge through pre-training or multi-task learning. Most of these approaches have focused on learning contextualized word representations with task-specific layers on top and have shown that syntactic knowledge can improve parsing (Zhou et al., 2020), semantic role labelling (Swayamdipta et al., 2018; Zhou et al., 2020), coreference resolution (Swayamdipta et al., 2018), grammatical error detection (Zhang et al., 2022) and relation extraction (Bassignana et al., 2023). Because these works focus on encoder-only models, they cannot be directly applied to sequence-to-sequence tasks.

In the context of sequence-to-sequence models, Xu et al. (2020) focus on broad-coverage semantic parsing and explore pre-training on multiple tasks including constituency parsing with linearized trees. Finally, Mulligan et al. (2021) present proof-of-concept experiments in which they show that multi-task learning of syntactic transformations can provide a bias towards hierarchical generalizations when data with a hierarchical structure is provided for the auxiliary tasks. In contrast to our work, they consider a setup with training from scratch using multi-task learning rather than pre-training. They only use three manually selected syntactic transformations and focus entirely on synthetic data.

To our knowledge, we are the first to explore pre-training with a large space of synthetic transformations of syntax trees. In addition, rather than using an atomic and unstructured task id to distinguish different tasks (Johnson et al., 2017; Xu et al., 2020; Mulligan et al., 2021), we provide the model with an explicit description of the transformation.

**Structural generalization** Several different approaches have been taken in recent works to improve the structural generalization of neural network models. One major line of research (Liu et al., 2021; Kim, 2021; Weißenhorn et al., 2022; Cazzaro et al., 2023; Lindemann et al., 2023a; Petit et al., 2023) has proposed a range of specialized architectures that have structural inductive biases by design. While very effective, these approaches tend to be difficult to train if the 'correct' task-specific syntactic analyses or alignments are not available, necessitating often complex and computationally expensive training algorithms. Since these approaches are also typically tailored to one or a few related tasks, architectures have to be redesigned when a new kind of task is considered.

Other works have explored data augmentation to improve structural generalization (Andreas, 2020; Qiu et al., 2022; Yang et al., 2022; Li et al., 2023b; Yao and Koller, 2024; Cazzaro et al., 2024). Because data augmentation is task-specific, it needs to be repeated and potentially also adapted to every new task. Data augmentation inherently risks introducing errors and noise to the training data. In contrast, our approach pre-trains a model once to perform syntactic transformations and can then be fine-tuned for different downstream tasks.

## 3 Strengthening Structural Inductive Bias

Standard pre-training objectives, e.g. with denoising objectives (Raffel et al., 2020), encourage models to acquire syntactic knowledge but provide little information about syntactic *transformations*, which are central to many syntactic and semantic seq2seq tasks. Our research hypothesis is that intermediate pre-training to perform transformations of syntax trees encourages the model to (i) strengthen its representations of the syntactic categories to which transformations can be applied (e.g. subjects, objects) and (ii) acquire *reusable* dynamics of transformations that are useful for downstream applications. By providing an explicit description of the transformation as a prefix, different transformations that the model has learned during pre-training can be 'activated' by the right choice of prefix. For this reason, we also fine-tune the model with a prefix of tunable embeddings to make it easy to leverage these transformations on downstream tasks similar to SIP (Lindemann et al., 2023b).

In addition to learning about transformations of trees, we also want the model to incorporate knowledge about the syntax of the underlying language (i.e. English, in this case). Hence, we do not provide syntax trees to the model during training, which also enables us to perform inference and fine-tuning without a parser.

### 3.1 Syntactic Transformations

Our goal in designing the transformations is to create a family of syntactic transformations which resemble a broad class of real downstream transformations. We base our syntactic transformations on Universal Dependency trees (de Marneffe et al., 2021), and provide an overview in Fig. 2. Each transformation is fully specified by a set of *edge-wise transformations* that assign a binary string operation (e.g. BRACKET) to a dependency relation (e.g. NSUBJ).

Applying a syntactic transformation to a dependency tree is a three-step process: First, we *unfold* the dependency tree into a binary 'phrase-structure'-like tree, where the dependency labels act as labels of the internal nodes.[2] This is neces-

[2]Related conversions from dependency to phrase structure trees have been explored in Xia and Palmer (2001).

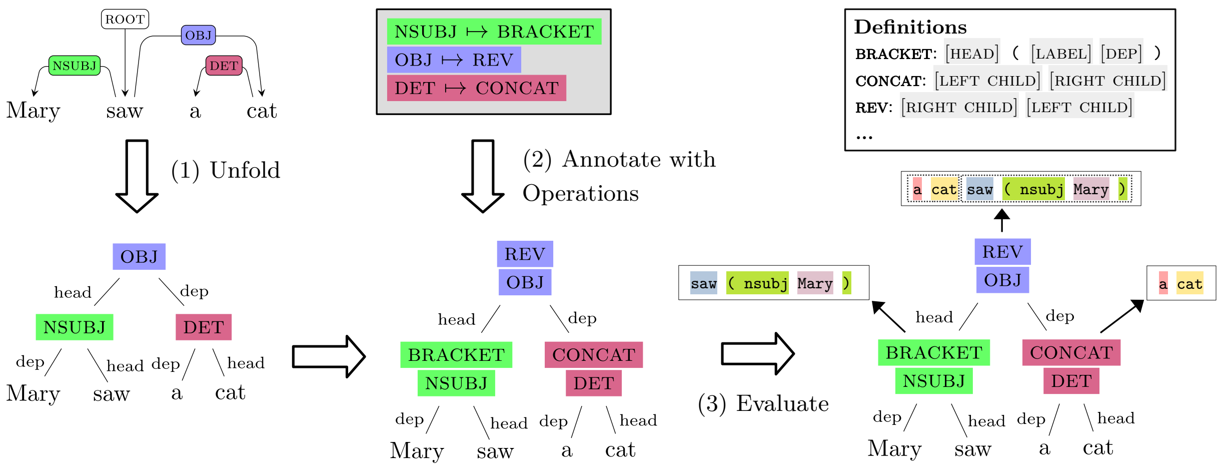

Figure 2: Our procedure of applying a syntactic transformation specified as edgewise transformations (grey box): (1) recursively unfolding a dependency tree into a binary tree where dependency labels serve as labels of internal nodes, (2) annotation dependency relations with edgewise transformations, (3), recursive evaluation of the edgewise transformations with partial results shown.

| Name | Definition | Example |
|---|---|---|
| CONCAT | LEFT CHILD RIGHT CHILD | Mary saw a cat |
| REV | RIGHT CHILD LEFT CHILD | a cat Mary saw |
| BRACKET | HEAD ( LABEL DEP ) | Mary saw ( obj a cat ) |
| BR-INVERT | DEP ( LABEL by HEAD ) | a cat ( obj by Mary saw ) |
| BRACKET-2 | ( HEAD LABEL DEP ) | ( Mary saw obj a cat ) |
| TRIPLE | HEAD ( HEAD.LEMMA LABEL DEP.LEMMA ) DEP | Mary saw ( see obj cat ) a cat |
| IGNORE-DEP | HEAD | Mary saw |

Table 1: General overview of the operations we use. We show an example transformation for the sentence *Mary saw a cat* where `HEAD = Mary saw` and `DEP = a cat`. HEAD.LEMMA (DEP.LEMMA) refers to the lemma of the head (dependent) that the node in question was unfolded from (in the example: saw $\xrightarrow{\text{obj}}$ cat). See Table A.1 for a full list of operations, including variants of those shown here.

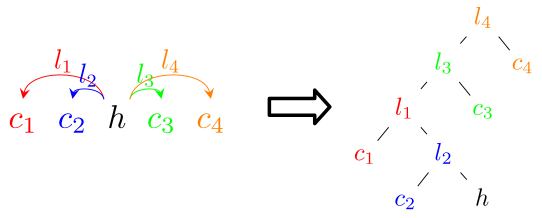

Figure 3: Unfolding a head $h$ and its children.

sary because all our operations are binary and we need a binary tree along which we can evaluate the operations. Second, we annotate the dependency labels with the corresponding operations according to the edge-wise transformations. Finally, we recursively evaluate each operation in the resulting expression tree, yielding a single output string.

Unfolding replaces a head and its dependents with a binarized tree, as shown in Fig. 3. This procedure is applied bottom-up to all nodes in the tree. For example, the dependency subtree of 'a cat' unfolds to the tree DET$(a, cat)$, after which 'saw' is unfolded, leading to the final unfolded result in Fig. 2. Unfolding a node without children (e.g. 'Mary') simply retains that node.

In order to have a wide range of syntactic transformations, we design an inventory of 14 operations to cover many potentially useful transformations for downstream tasks (see Table 1 for a general overview, and Table A.1 for the full list). Note that assigning the CONCAT operation to all dependency relations results in an output that is identical to the input if the dependency tree is projective.

Our syntactic transformations could in principle also be implemented with synchronous grammars (Lewis and Stearns, 1968; Chiang, 2007). In contrast to our transformations, the rules of synchronous grammars generate not only the output but also the input. As a result, synchronous grammars would yield a much more verbose representation with rules that apply to specific words only, potentially harming generality of the representations learned by the model.

### 3.2 Intermediate pre-training

During intermediate pre-training, the model is given a sentence and a set of edgewise transformations that determine the overall transformation. The objective is to predict what the transformation does to the parse tree of the sentence. The input to the Transformer is a sequence of vectors from $\mathbb{R}^d$, which consist of a prefix that represents the edgewise transformations and a suffix comprised of the embeddings of the input tokens:

$$\underbrace{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k}_{\text{Transformation}}, \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Sentence}}$$

Each $\mathbf{h}_i$ encodes an edge-wise transformation $R \mapsto f$ by simple addition of embeddings:

$$\mathbf{h}_i = \text{EMBED}_{\text{Label}}(R) + \text{EMBED}_{\text{Transformation}}(f)$$

The training objective is the log likelihood of the correct output of the transformation, and we start from T5-base (Raffel et al., 2020) that has already been pretrained. Note that the dependency tree is *not* provided to the model to encourage it to reuse and strengthen its syntactic knowledge.

We also want to preserve the existing (syntactic) knowledge of the pre-trained T5 model, e.g. to make it easy to insert the right auxiliary verb form when transforming a sentence from active to passive (see Fig. 1). To help preserve this, our second pre-training objective is the span-denoising objective that T5 was originally pre-trained with. We train the model by alternating between gradient descent steps on the two objectives.

**Data generation** We construct random syntactic transformations for a small fraction of the C4 corpus, which T5 was originally pre-trained on. We tag, parse and lemmatize 2.1 million sentences with a total of around 39 million word forms using trankit (Nguyen et al., 2021). We create two random transformations per parsed sentence, resulting in approximately 4.2 million pre-training instances.

To construct a random syntactic transformation for a given sentence, we sample dependency relations present in that sentence and some additional dependency relations that are *not* present in the sentence to a maximum total of 20 relations. We uniformly sample an operation for each relation to create edgewise transformations. Relations that are not chosen by sampling are implicitly assigned the operation CONCAT. While the relations that are not present in the sentence have no bearing on the output of the transformation, we include them in the description to expose the model to a more general description that applies to a broader range of sentences.

### 3.3 Fine-tuning

After pre-training, we apply our model to different downstream tasks via fine-tuning. Mirroring the pre-training, we replace the transformation encoding with a sequence of tunable embeddings. That is, the input to the model is a sequence of vectors:

$$\underbrace{\mathbf{h}'_1, \mathbf{h}'_2, \ldots, \mathbf{h}'_k}_{\text{Tunable embeddings}}, \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Sentence}}$$

where $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n$ are the embeddings of the input tokens, $\mathbf{h}'_i \in \mathbb{R}^d$ are the tunable embeddings and $k$ is a hyperparameter. The embeddings $\mathbf{h}'_i$ are initialized to the average of the encoding of multiple transformations from the pre-training phase. Because the tuneable embeddings are trained on the downstream task, they can be used to 'activate' transformations that help with the particular downstream task. We fine-tune all model parameters and use a higher learning rate for the prefix.

## 4 Evaluation

We evaluate on syntactic and semantic tasks for which a structural inductive bias should be helpful. Specifically, we consider learning from a small amount of task-specific data (few-shot learning) and structural generalization outside of the training distribution to unseen combinations of known phrases, novel syntactic phenomena and deeper recursion than seen during training.

### 4.1 Baselines

For a fair comparison, we compare our method (STEP, for **S**yntactic **T**ransformation **E**nhanced **P**re-training) with fine-tuning other seq2seq models based on T5-base (Raffel et al., 2020) that were further pre-trained on the parsed corpus (Section 3.2) in different ways:

**T5+Dep Parse** is pre-trained to predict a linearized dependency tree of the input, e.g. *Mary saw a cat* → `( saw nsubj Mary obj ( cat det a ) )`. Hence, this model incorporates syntactic information about English dependency trees but has limited exposure to how this information can be used other than to produce a parse tree.

**Simple STEP** is a simplified version of STEP, where we always assign the same edgewise transfor-

| Task | Model | Acc ↑ | BLEU ↑ | TER ↓ |
|---|---|---|---|---|
| Verb emphasis | T5 | 3.5 | 41.7 | 46.7 |
| | T5+Dep Parse | 3.3 | 40.1 | 48.2 |
| | Simple STEP | **3.6** | 40.5 | 47.0 |
| | STEP | 3.4 | **41.8** | **45.6** |
| Adj. emphasis | T5 | 7.3 | 47.6 | 38.3 |
| | T5+Dep Parse | 7.7 | 45.8 | 40.4 |
| | Simple STEP | 9.8 | 48.3 | 37.5 |
| | STEP | **10.9** | **52.3** | **33.5** |
| Passivization | T5 | 40.2 | 73.7 | 18.3 |
| | T5+Dep Parse | 45.0 | 76.8 | 15.5 |
| | Simple STEP | 46.8 | 78.4 | 13.6 |
| | STEP | **57.9** | **84.8** | **8.4** |

Table 2: Evaluation on 100-shot **syntactic transformation** tasks. We report averages of 10 draws of 100 training examples each.

mation to *all* dependency relations. Consequently, the number of possible syntactic transformations is exactly the number of binary string operations we define (Table A.1). We remove IGNORE-DEP because it would result in an output string with a single token. We use a special token in the prefix to indicate which transformation should be applied.

Analogously to STEP, the models above were pre-trained with their specific pre-training objective and the original span denoising objective of T5.

## 4.2 Syntactic Tasks

We first evaluate if our synthetic transformations transfer to realistic syntactic transformations. In particular, we focus on few-shot scenarios.

We evaluate on three structural transformations that Lyu et al. (2021) identified as challenging because only several hundreds of training examples are available: passivization (Fig. 1), emphasis of a designated adjective[3] and emphasis of a designated verb[4]. We consider a more challenging version of this with only 100 training examples.

We report results in Table 2 using exact match accuracy, BLEU (Papineni et al., 2002) and TER (Snover et al., 2006), a normalized edit distance. Using dependency parsing as the intermediate pre-training task (T5+Dep Parse) is already beneficial for passivization but somewhat deteriorates performance on adjective emphasis both in terms of BLEU and TER. Simple STEP improves on this with small gains on both adjective emphasis and small additional improvements for passivization. STEP performs best, outperforming the baselines by a sizable margin of 3.5 and 6 points BLEU on the adjective emphasis and passivization tasks. However, STEP and T5 perform similarly on the verb emphasis task, and we hypothesize STEP has difficulties reusing the transformations acquired during pre-training (see also Section 5).

| Model | Acc ↑ | F ↑ |
|---|---|---|
| T5 | 34.4±0.8 | 87.4±0.6 |
| T5+Dep Parse | 39.9±2.1 | 90.0±0.6 |
| Simple STEP | 45.3±2.0 | 90.6±0.6 |
| STEP | **53.8**±2.1 | **93.2**±0.5 |

Table 3: Means and standard deviations on **chunking** across 5 random draws of 100 training examples. Accuracy is exact match, i.e. predicting *all* chunks correctly.

**Chunking** We also evaluate on chunking (Tjong Kim Sang and Buchholz, 2000) phrased as a seq2seq task.[5] Different variants of chunking play an important role in information extraction (Dong et al., 2023), which often has to rely on small domain-specific corpora (Bassignana and Plank, 2022). Few-shot learning of chunking is hence relevant and particularly interesting in our setup because it requires models to predict phrase categories (e.g. NPs) that do not exist in our pre-training approach based on dependency trees.

We report results in Table 3. While using parsing as intermediate pre-training is already helpful in comparison to T5, STEP improves accuracy even further and outperforms T5 by almost 20 percentage points for exact match accuracy. Simple STEP also shows some improvements over T5+Dep Parse but is again outperformed by STEP.

Overall, this shows that STEP strengthens the inductive bias for realistic syntactic transformations. The improvements of STEP over T5 cannot be attributed alone to the prediction of dependency trees during pre-training as T5+Dep Parse performs worse. Pre-training the model with a narrow set of transformations (Simple STEP) is not as effective as a large set of transformations with explicit descriptions. We hypothesize that the improvements of STEP can be attributed partly to the reusability of the transformations during fine-tuning, which we analyze in Section 5.

[3]*The $\underline{\text{French}}$ analysis goes further* → *The analysis that goes further is French*

[4]*corporate profits may also $\underline{\text{dip}}$ initially* → *the dipping of corporate profits may also $\underline{\text{happen}}$ initially*

[5]*The chairman promised Mr. Stone a decision* → *(NP The chairman) (VP promised) (NP Mr. Stone) (NP a decision)*

| **Model** | **Modifiers** | **Novel Gaps** | **Wh-Questions** | **Recursion** | **Overall** |
|---|---|---|---|---|---|
| *AM-Parser** | 69.6 | 20.7 | 57.0 | **99.9** | 70.8±4.3 |
| T5 | 79.5±2.5 | 81.4±6.2 | 82.4±2.3 | 71.1±1.2 | 77.6±1.4 |
| T5+Dep Parse | 82.8±3.1 | **86.8**±6.6 | 78.8±4.4 | 72.2±2.0 | 78.3±2.1 |
| Simple STEP | **88.6**±0.4 | 44.8±12.6 | 84.2±2.1 | 79.0±1.6 | 78.8±1.9 |
| STEP | 87.5±0.4 | 47.6±14.1 | **84.8**±4.2 | 79.8±0.9 | **79.3**±2.3 |

Table 4: Structural generalization on the variable-free meaning representation of **SLOG** based on 10 random seeds. Specialized architectures are represented with italics. * The AM-Parser uses a semantically more expressive meaning representation formalism based on graphs.

| **Model** | **iid** | **Length** |
|---|---|---|
| *Tag & Permute* | 76.6±1.7 | **41.4**±13.5 |
| T5 | **85.5**±1.4 | 32.0±4.4 |
| T5+Dep Parse | 84.9±0.4 | 27.5±2.0 |
| Simple STEP | 82.9±1.0 | 30.4±1.5 |
| STEP | 84.2±1.7 | 38.4±2.7 |

Table 5: Means and standard deviations of model accuracy for semantic parsing on **ATIS** for 5 random seeds.

## 4.3 Semantic Tasks

Semantic parsing, i.e. constructing a logical form from a sentence, can be seen as a particular transformation of the syntactic structure (Montague, 1970). Hence, we expect an inductive bias for syntactic transformations to be helpful for semantic parsing, particularly for *structural generalization*, i.e. extrapolation to unseen combinations of phrases, longer examples and deeper recursion.

**SLOG** (Li et al., 2023a) is a synthetic benchmark that tests models on 17 different structural generalizations grouped into 4 categories: using modifiers in novel positions (e.g. PPs only modify objects during training but modify subjects at test time), novel gap positions (e.g. wh-question for an indirect object, with the training data covering wh-questions for subjects and objects), wh-questions in novel syntactic contexts (e.g. wh-questions combined with passive instead of active voice) and recursion (e.g. deeper PP recursion).

We report aggregated results in Table 4 and results for all 17 generalization cases in Table B.2. Overall, STEP performs best but performance on the different categories varies considerably between the approaches. STEP and Simple STEP outperform T5 on all but one category, with considerable margins for the novel modifier positions and unseen recursion depths. However, they underperform in the case of the novel gap positions. T5+Dep Parse performs more similarly to T5 with typically modest improvements across the categories.

We also compare with the AM-Parser (Groschwitz et al., 2018; Weißenhorn et al., 2022), a specialized approach for semantic parsing. It performs worse than the seq2seq models on most categories, except for recursion, where it achieves close to perfect accuracy. Here, STEP reduces the gap between the more general seq2seq models and the specialized AM-Parser. Interestingly, both STEP and Simple STEP improve over T5 on generalization to center embedding of depth 5 or more by 8 and 14 percentage points respectively even though there is no evidence of center embedding of depth two or more in our parsed corpus (Table B.2 and Fig. B.2).

**ATIS** (Dahl et al., 1994) is a semantic parsing dataset with questions about a flight database annotated with executable logical forms. We follow previous work in using the variable-free FunQL version (Guo et al., 2020). However, we found that the order of the conjuncts in the logical form tends to be somewhat unsystematic and often does not correspond to the linear order in the question. Hence, we use a pre-processing step to re-order conjuncts based on automatic alignments (see Appendix A.1). We evaluate in two setups: (i) the standard iid split and (ii) a length split, where a model is shown logical forms with up to three conjuncts during training and has to generalize to sentences that require four or more conjuncts in the logical form.

Results are shown in Table 5. Tag & Permute (Lindemann et al., 2023a) is a specialized architecture for semantic parsing and is currently state-of-the-art on the length split. STEP performs best among the non-specialized architectures on the length split, narrowing the gap to the specialized model. Interestingly, T5+Dep Parse and Simple STEP perform somewhat worse than plain T5.

## 5 Analysis

Our research hypothesis is that our intermediate pre-training encourages the model to acquire reusable dynamics of syntactic transformations that can be leveraged during fine-tuning. In this section, we analyze the representations used by our model after its intermediate pre-training, and to what degree they are reused during fine-tuning.

### 5.1 Analysis of Pre-Trained Model

We first investigate how the model processes the transformation encoded in the prefix. The model has to attend to the prefix to gather information about which edgewise transformation needs to be applied to which input token. We call an attention head a *transformation look-up head* if it consistently attends to the prefix.

We find that some transformation look-up heads are interpretable and follow syntactic patterns. For example, when head 6 in layer 10 computes the attention distribution for a token that is an object in the sentence (i.e. *cat* in Fig. 1), it focuses the attention on the edgewise transformation that describes how to process objects (i.e. OBJ $\mapsto$ REV).

**Identifying interpretable look-up heads** We consider each attention head $H$ and dependency relation $R$ separately. For a sample of sentences with corresponding transformations, we count how many times the following conditions are true: (i) the instance has an edgewise transformation involving $R$, (ii) a token $x_i$ has an incoming edge labelled $R$ and (iii) $H$ focuses at least 50% of its attention from $x_i$ on a single position $j$. If in over 70% of cases, position $j$ refers to the edgewise transformation of $R$ then we call $H$ a transformation look-up head for the dependency relation $R$.

We find that there are often multiple transformation look-up heads per dependency relation. For example, we identify 7 look-up heads for amod. These interpretable heads are typically located in the mid or higher layers (see Fig. B.3), which is expected because the model first needs to identify the syntactic role each token has.

**Intervening on look-up heads** Next, we verify that the transformation look-up heads we identified contribute to the model prediction with an interventional analysis. We evaluate the role of the transformation look-up heads separately for different dependency relations: if the heads $H_1, H_2, \dots H_n$ play an important role in performing transformations for dependency relation $R$, then masking all of them should drop accuracy for instances with an edgewise transformation for $R$. As a comparison, we also evaluate (i) masking $n$ randomly chosen heads, and (ii) masking $n$ randomly chosen heads that are transformation look-up heads, but not for $R$. Since the look-up heads can also have other functions within the model, we only mask out the attention to the prefix. When masking randomly selected attention heads, we ensure comparability by masking a random subset of tokens equal to the length of the prefix.

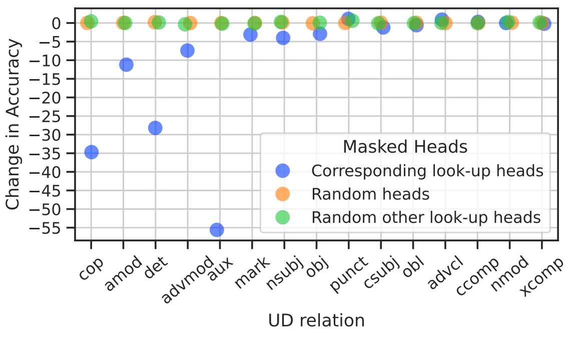

Figure 4: Change in accuracy of predicting the output of edgewise transformations when masking different attention heads. We show accuracy relative to no masking. The effect of masking random heads and random other look-up heads tend to be very similar, leading to overlap in the plot.

We show the results in Fig. 4. Masking transformation look-up heads reduces accuracy for many dependency relations while masking other transformation look-up heads or random heads has very little impact. This provides evidence that the identified heads play an important role within the model. For some relations (e.g. punct, advcl, nmod), masking the respective look-up heads does not reduce accuracy, suggesting that responsibility for these relations is more spread out through the network.

### 5.2 Analysis of Fine-Tuned Models

How does a model pre-trained with STEP learn during fine-tuning? We hypothesize that the pre-training provides a scaffolding, which finetuning can build upon. In particular, we expect that aspects of the downstream task that can be expressed with our transformations to be captured in the same way as during pre-training, i.e. with the prefix and with the transformation look-up heads.

**Masking look-up heads after fine-tuning** To test this hypothesis, we create 10 new synthetic

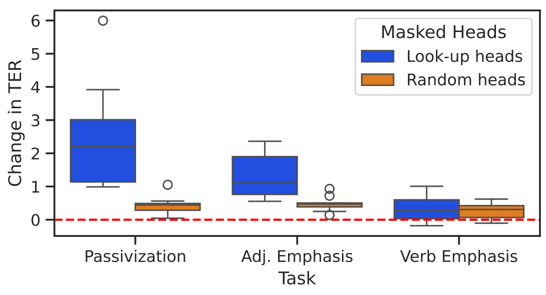

Figure 5: Effect of masking look-up heads of models that have been fine-tuned on downstream syntactic tasks. For each task, we show the distribution for the 10 fine-tuned models from Section 4.2.

transformation tasks with 5 edgewise transformations each and fine-tune the model (Section 3.3). Then, we take the attention heads we identified in Section 5.1 and repeat the masking intervention, i.e. we mask the attention to the prefix of all look-up heads for the dependency relations involved in the edgewise transformations.

Masking the look-up heads of the dependency relations involved in the transformations leads to an average drop in accuracy of 30 percentage points (see also Fig. B.1), whereas masking random look-up heads reduces accuracy by less than one point.

**Reading off transformations from fine-tuned prefix** The strong impact of masking the look-up heads in the fine-tuned model suggests that it uses the tunable prefix to encode task-specific information about which edgewise transformation to apply. To gain more insight into this, we try to *extract* edgewise transformations from the fine-tuned prefix: For each vector $\mathbf{h}'$ in the prefix, we find the edgewise transformation whose embedding is closest to $\mathbf{h}'$ in terms of cosine similarity. In this manner, we can read off a candidate for the transformation which the model might be using under the hood and compare it to the correct transformation that generated the synthetic data. We find that the edgewise transformations extracted in this way agree with the gold edgewise transformations with an average F-score of $\approx 77$. This provides evidence for the hypothesis that the model re-uses the transformations learned during pre-training and 'activates' them with the prefix.

**Fine-tuning on realistic transformations** Finally, we investigate the role of transformation look-up heads in models fine-tuned on realistic syntactic transformations outside of the pre-training distribution (see Section 4.2). Since there are no ground truth edgewise transformations in this case, we mask the attention to the prefix of *all* transformation look-up heads and compare with masking an equal number of random heads. Fig. 5 shows that masking the transformation look-up heads deteriorates outputs more than masking random heads for passivization and the adjective emphasis task. However, results are comparable for verb emphasis. This is in line with our findings that STEP improves over T5 for passivization and adjective emphasis but not for verb emphasis, and suggests that the lack of improvement for the verb emphasis task could be due to difficulties in reusing the transformations seen during intermediate pre-training.

## 6 Conclusion

We propose a new method of strengthening the structural inductive bias of a Transformer by pre-training the model to perform syntactic transformations based on dependency trees. We show that this results in a better few-shot performance for syntax-dependent seq2seq tasks, and also improves structural generalization for semantic parsing.

Analysis of the pre-trained model shows that it uses attention heads to track what transformation needs to be applied to which input token, and that these heads tend to follow syntactic patterns. In addition, we find that fine-tuning re-uses these attention heads, suggesting that the model can leverage the transformations acquired during pre-training.

## Limitations

The structural inductive bias that is emphasized by our intermediate pre-training depends on the inventory of operations. Due to the computational cost of pre-training, we did not systematically explore which set of operations performs best, or which operations do not provide much benefit and could be omitted.

In this work, we focus on a moderately sized encoder-decoder model (T5) and do not investigate large decoder-only models. However, we conjecture that the decoder-only architecture is compatible with our approach. While larger models trained on more data might have stronger syntactic capabilities and therefore likely have less of a need to address issues of syntactic generalization overall, we think our approach could be helpful in addressing remaining issues with structural generalization.

Finally, our analysis focuses on the encoder on the Transformer, and on the transformation look-up heads in particular. However, applying a transformation also requires appropriate mechanisms in the decoder, and the picture of how this works internally remains much less clear.

## Acknowledgements

ML is supported by the UKRI Centre for Doctoral Training in Natural Language Processing, funded by the UKRI (grant EP/S022481/1), the University of Edinburgh, School of Informatics and School of Philosophy, Psychology & Language Sciences, and a grant from Huawei Technologies. IT is supported by the Dutch National Science Foundation (NWO Vici VI.C.212.053).

## References

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.

Elisa Bassignana, Filip Ginter, Sampo Pyysalo, Rob van der Goot, and Barbara Plank. 2023. Silver syntax pre-training for cross-domain relation extraction. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6984–6993, Toronto, Canada. Association for Computational Linguistics.

Elisa Bassignana and Barbara Plank. 2022. CrossRE: A cross-domain dataset for relation extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3592–3604, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Francesco Cazzaro, Davide Locatelli, and Ariadna Quattoni. 2024. Align and augment: Generative data augmentation for compositional generalization. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 369–383, St. Julian's, Malta. Association for Computational Linguistics.

Francesco Cazzaro, Davide Locatelli, Ariadna Quattoni, and Xavier Carreras. 2023. Translate first reorder later: Leveraging monotonicity in semantic parsing. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 227–238, Dubrovnik, Croatia. Association for Computational Linguistics.

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

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.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Kuicai Dong, Aixin Sun, Jung-jae Kim, and Xiaoli Li. 2023. Open information extraction via chunks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15390–15404, Singapore. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

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

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

Jiaqi Guo, Qian Liu, Jian-Guang Lou, Zhenwen Li, Xueqing Liu, Tao Xie, and Ting Liu. 2020. Benchmarking meaning representations in neural semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1520–1540, Online. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. 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 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

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 *International Conference on Learning Representations*.

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*, volume 34, pages 26302–26317. Curran Associates, Inc.

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

Philip M Lewis and Richard Edwin Stearns. 1968. Syntax-directed transduction. *Journal of the ACM (JACM)*, 15(3):465–488.

Bingzhi Li, Lucia Donatelli, Alexander Koller, Tal Linzen, Yuekun Yao, and Najoung Kim. 2023a. SLOG: A structural generalization benchmark for semantic parsing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3213–3232, Singapore. Association for Computational Linguistics.

Zhaoyi Li, Ying Wei, and Defu Lian. 2023b. Learning to substitute spans towards improving compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2791–2811, Toronto, Canada. Association for Computational Linguistics.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023a. Compositional generalization without trees using multiset tagging and latent permutations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14488–14506, Toronto, Canada. Association for Computational Linguistics.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023b. Injecting a structural inductive bias into a seq2seq model by simulation. *arXiv preprint arXiv:2310.00796*.

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.

Yiwei Lyu, Paul Pu Liang, Hai Pham, Eduard Hovy, Barnabás Póczos, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2021. StylePTB: A compositional benchmark for fine-grained controllable text style transfer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2116–2138, Online. Association for Computational Linguistics.

R Thomas McCoy and Thomas L Griffiths. 2023. Modeling rapid language learning by distilling bayesian priors into artificial neural networks. *arXiv preprint arXiv:2305.14701*.

Richard Montague. 1970. English as a formal language. In Bruno Visentini, editor, *Linguaggi nella societa e nella tecnica*, pages 188–221. Edizioni di Communita.

Aaron Mueller, Robert Frank, Tal Linzen, Luheng Wang, and Sebastian Schuster. 2022. Coloring the blank slate: Pre-training imparts a hierarchical inductive bias to sequence-to-sequence models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1352–1368, Dublin, Ireland. Association for Computational Linguistics.

Karl Mulligan, Robert Frank, and Tal Linzen. 2021. Structure here, bias there: Hierarchical generalization by jointly learning syntactic transformations. In *Proceedings of the Society for Computation in Linguistics 2021*, pages 125–135, Online. Association for Computational Linguistics.

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 80–90, Online. Association for Computational Linguistics.

Karel Oliva. 1988. Syntactic functions in GPSG. In *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*.

Isabel Papadimitriou and Dan Jurafsky. 2023. Injecting structural hints: Using language models to study inductive biases in language learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8402–8413, Singapore. Association for Computational Linguistics.

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

*40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Alban Petit, Caio Corro, and François Yvon. 2023. Structural generalization in COGS: Supertagging is (almost) all you need. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1089–1101, Singapore. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4341–4362, Seattle, United States. 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(140):1–67.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. 2018. Syntactic scaffolds for semantic structures. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3772–3782, Brussels, Belgium. Association for Computational Linguistics.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*.

Pia Weißenhorn, Lucia Donatelli, and Alexander Koller. 2022. Compositional generalization with a broad-coverage semantic parser. In *Proceedings of the 11th Joint Conference on Lexical and Computational Semantics*, pages 44–54, Seattle, Washington. Association for Computational Linguistics.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020. Improving AMR parsing with sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2501–2511, Online. Association for Computational Linguistics.

Jingfeng Yang, Le Zhang, and Diyi Yang. 2022. SUBS: Subtree substitution for compositional semantic parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 169–174, Seattle, United States. Association for Computational Linguistics.

Yuekun Yao and Alexander Koller. 2022. Structural generalization is hard for sequence-to-sequence models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yuekun Yao and Alexander Koller. 2024. Simple and effective data augmentation for compositional generalization. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 434–449, Mexico City, Mexico. Association for Computational Linguistics.

Shuai Zhang, Wang Lijie, Xinyan Xiao, and Hua Wu. 2022. Syntax-guided contrastive learning for pre-trained language model. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2430–2440, Dublin, Ireland. Association for Computational Linguistics.

Junru Zhou, Zhuosheng Zhang, Hai Zhao, and Shuailiang Zhang. 2020. LIMIT-BERT : Linguistics informed multi-task BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4450–4461, Online. Association for Computational Linguistics.

## A Additional Details

### A.1 Pre-processing

**SLOG** For SLOG, we remove `nmod.` from the logical forms to shorten them and to avoid giving models pre-trained with syntax trees a potential advantage simply because the downstream logical form uses a similar token to a dependency label. Hence, the logical form for 'Isabella forwarded a box on a tree to Emma.' becomes `forward ( agent = Isabella , theme = box ( on = tree ) , recipient = Emma )` after pre-processing with the original one being `forward ( agent = Isabella , theme = box ( nmod . on = tree ) , recipient = Emma )`.

**ATIS** We train an IBM-1 alignment model on the pairs of sentences and logical forms, and then sort the conjuncts by their sum-total expected alignment: let $A_{i,j}$ be the posterior probability that the input token at position $i$ is aligned to output the output at position $j$. Let $C$ be the set of output token positions belonging to a conjunct. We then let $A(C) = \sum_{j \in C} \sum_i A_{i,j} \cdot i$. We repeat this for every conjunct and then sort them.

## A.2 Experimental setup

**Syntactic Transformations and Chunking** We evaluate in a few-shot scenario with only 100 training examples, and do not assume access to a development set. For this reason, we don't tune hyperparameters and fine-tune for a fixed number of epochs. As performance can differ from checkpoint to checkpoint, for each run, during the last 10 epochs, we evaluate on the test set and use the average result as the performance of that run.

For adjective emphasis, verb emphasis and passivization, we use all examples besides the 100 training examples as test set, i.e. 2635 test examples for passivization, 596 for adjective emphasis, and 1101 for verb emphasis. For chunking, we use the test set from Tjong Kim Sang and Buchholz (2000).

**ATIS** We follow Lindemann et al. (2023a) in using the development set to select the best epoch based on the accuracy metric, which is also used on the test set (rather than loss).

**SLOG** SLOG does not have an out-of-distribution development set, so we train for a fixed number of epochs that was determined by the hyperparameter search (see Appendix A.3).

**Identifying look-up heads** We use a sample of 1000 unseen sentences from the C4 corpus along with randomly generated transformations as described in Section 3.1 to identify interpretable look-up heads.

**Intervening on look-up heads** Since we want to evaluate the impact of look-up heads for particular dependency relations, we create a dataset with 1000 examples of transformations per dependency relation. To avoid confounding factors, each instance has only a single edgewise transformation (for the specific dependency relation).

When we mask random attention heads or random look-up heads, it is computationally too expensive to do this for all possible attention heads and we approximate this with a Monte Carlo estimate: we select random heads 20 times and take the average of the results.

**Analysis of fine-tuned models on synthetic data** When generating synthetic downstream tasks, we exclude the CONCAT operation for edgewise transformations. We take a sample of 5000 sentences from our parsed corpus and randomly divide it into an 80/20 train/test split. We use a prefix of tunable embeddings of the same length as the ground truth, i.e. we set it to a length of 5. When masking random (look-up) heads, we repeat this 50 times to estimate the expected change in accuracy.

## A.3 Hyperparameters & Hardware

**Pre-training** When generating pre-training data for STEP, we only use sentences with 90 or less tokens (in terms of the T5 tokenizer) and exclude any instances with outputs of 180 T5 tokens or more. However, we do not impose a limit on the length of the output for our baselines (T5+Dep Parse and Simple STEP) because it would remove too much of the pre-training data. We use Adafactor for our intermediate pre-training with a learning rate of $3 \times 10^{-4}$ and without warm-up, and a batch size of 80 (for STEP), 30 (for Simple STEP) and 48 (for T5+Dep Parse). We maintain separate optimizers for the main objective (e.g. predicting the transformation) and the original span-denoising objective. We train for a single epoch, except for T5+Dep Parse, which we train for two epochs. This is because STEP and Simple STEP have two instances with syntactic transformations per parsed sentence but T5+Dep Parse only has a single one. For the denoising objective, we impose a limit of 80 tokens per instance (truncating longer instances) and use a batch size of 50.

**Fine-tuning** During fine-tuning, the main hyperparameters are the learning rates. We use Adafactor for fine-tuning using a learning rate of $1 \times 10^{-4}$. For the prefix of STEP, we use a learning rate of

| Name | Definition | | Example |
|---|---|---|---|
| CONCAT | LEFT CHILD RIGHT CHILD | | Mary saw a cat |
| REV | RIGHT CHILD LEFT CHILD | | a cat Mary saw |
| CONCAT-REL | LEFT CHILD LABEL RIGHT CHILD | | Mary saw obj a cat |
| REVL-REL | RIGHT CHILD LABEL LEFT CHILD | | a cat obj Mary saw |
| BRACKET | HEAD ( LABEL DEP ) | | Mary saw ( obj a cat ) |
| BR-INVERT | DEP ( LABEL by HEAD ) | | a cat ( obj by Mary saw ) |
| BRACKET-2 | ( HEAD LABEL DEP ) | | ( Mary saw obj a cat ) |
| BRACKET-2-INV | ( DEP LABEL HEAD ) | | ( a cat obj Mary saw ) |
| BRACKET-3 | HEAD ( DEP ) | | Mary saw ( a cat ) |
| BRACKET-4 | HEAD LABEL ( DEP ) | | Mary saw obj ( a cat ) |
| BRACKET-5 | HEAD ( LABEL DEP ) | if head has no other BRACKET-5 arguments | Mary saw ( obj a cat ) |
| | HEAD ( LABEL DEP | if this is the first BRACKET-5 argument | |
| | HEAD , LABEL DEP ) | if this is the last BRACKET-5 argument | |
| | HEAD , LABEL DEP | else | |
| TRIPLE | HEAD ( HEAD.LEMMA LABEL DEP.LEMMA ) DEP | | Mary saw ( see obj cat ) a cat |
| TRIPLE-INV | HEAD ( DEP.LEMMA LABEL by HEAD.LEMMA ) DEP | | Mary saw ( cat obj by see ) a cat |
| IGNORE-DEP | HEAD | | Mary saw |

Table A.1: Full list of operations we use. We show an example transformation for the sentence *Mary saw a cat* where `HEAD = Mary saw` and `DEP = a cat`. HEAD.LEMMA (DEP.LEMMA) refers to the lemma of the head (dependent) that the edge in question was unfolded from (in the example: saw $\xrightarrow{\text{obj}}$ cat). BRACKET-5 essentially concatenates the results of all other BRACKET-5 children together using a comma as joining element, and surrounds this with one matching pair of brackets. If in the example, we had edgewise transformations NSUBJ $\mapsto$ BRACKET-5 and OBJ $\mapsto$ BRACKET-5, the output would be `saw ( nsubj Mary , obj a cat )`, similar to our linearization of dependency trees for T5+Dep Parse. Formally, we call a subtree an $\ell$ argument in the unfolded and annotated tree if it is a non-head child that is dominated by a node that is annotated with operation $\ell$. For example, in Fig. 2, the subtree corresponding to 'a cat' is a CONCAT argument.

10. These hyperparameters apply to all experiments and all models, except for SLOG, as described below:

**SLOG** We found that accuracy on SLOG was very sensitive to hyperparameters and used a hyperparameter selection strategy similar to that of Conklin et al. (2021) for COGS: we draw a sample of around 10% from the generalization data. We fixed one random seed and ran 10 randomly sampled hyperparameter configurations and selected the one with the highest accuracy that was most stable across the epochs. We then discarded that random seed and used different ones for fine-tuning the model. We sample the learning rate from LogUniform$[2 \times 10^{-6}, 1 \times 10^{-4}]$ and the batch size uniformly from $[24, 48, 72, 96, 120]$. STEP also has an additional learning rate for the prefix, which we sample from LogUniform$[0.1, 10]$ during the search. The chosen hyperparameters can be found in Table A.2.

| Model | Epochs | Batch size | LR | LR Prefix |
|---|---|---|---|---|
| T5 | 50 | 96 | 1.62E-05 | - |
| T5+ Dep Parse | 50 | 24 | 9.51E-05 | - |
| Simple STEP | 22 | 72 | 6.75E-05 | - |
| STEP | 15 | 48 | 1.30E-05 | 2.52 |

Table A.2: Hyperparameters used for **SLOG**. LR is learning rate.

**Hardware** All our experiments were run on Nvidia 2080TI or 1080TI GPUs. Pre-training STEP took around 30 hours. Since we used longer maximum sequence lengths for the baselines (see above), and had to decrease the physical batch size, training of the baselines took 50 (T5+Dep Parse) and 95 hours (Simple STEP).

**Number of parameters** T5-base has 222 million parameters. When we fine-tune STEP with a prefix of tunable embeddings, this adds 7860 parameters to that, which is an increase of 0.035 ‰.

## A.4 Evaluation metrics

We use SacreBLEU (Post, 2018) v2.3 to compute BLEU and TER. For the experiments with ATIS, we use the code of Lindemann et al. (2023a) for computing accuracy.

| Task | Model | Acc ↑ | BLEU ↑ | TER ↓ |
|---|---|---|---|---|
| Adj. emphasis | STEP | 10.9±1.0 | 52.3±0.7 | 33.5±0.6 |
| | Simple STEP | 9.8±0.8 | 48.3±0.8 | 37.5±0.9 |
| | T5 | 7.3±0.8 | 47.6±0.8 | 38.3±0.8 |
| | T5+Dep Parse | 7.7±0.8 | 45.8±0.9 | 40.4±1.1 |
| Passivization | STEP | 57.9±2.0 | 84.8±0.7 | 8.4±0.5 |
| | Simple STEP | 46.8±2.2 | 78.4±0.8 | 13.6±0.6 |
| | T5 | 40.2±1.7 | 73.7±0.8 | 18.3±0.7 |
| | T5+Dep Parse | 45.0±1.6 | 76.8±0.5 | 15.5±0.4 |
| Verb emphasis | STEP | 3.4±0.4 | 41.8±0.6 | 45.6±0.4 |
| | Simple STEP | 3.6±0.7 | 40.5±0.6 | 47.0±0.6 |
| | T5 | 3.5±0.4 | 41.7±0.5 | 46.7±0.4 |
| | T5+Dep Parse | 3.3±0.7 | 40.1±0.8 | 48.2±0.6 |

Table B.1: Evaluation on 100-shot **syntactic transformation** tasks. We report averages of 10 draws of 100 training examples each. We also include standard deviations on the results across the 10 runs.

**SLOG** Li et al. (2023a) argue for using semantic equivalence for evaluation but they focus on a variable-based formalism and use exact string match for evaluating the variable-free representation. We take semantic equivalence into account, in particular, the order of the children does not matter because the roles are represented in the logical form. Hence, `offer ( theme = donut , recipient = * turtle )` and `offer ( recipient = * turtle , theme = donut )` are equivalent. We achieve this by parsing the string representation into a tree and instead of a list of children we maintain a set of children, and then compare trees to evaluate accuracy.

## B Additional Results

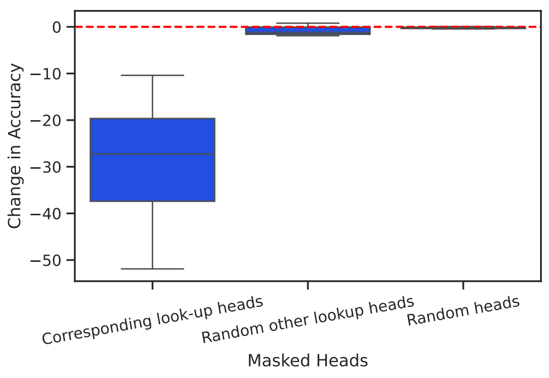

Figure B.1: Effect of masking look-up heads of models fine-tuned on synthetic tasks. The boxplot shows the distribution for 10 synthetic downstream tasks.

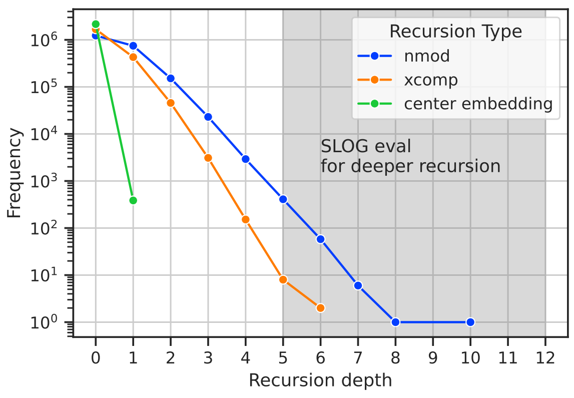

Figure B.2: Frequency of recursion depths in our parsed corpus (Section 3.2) according to the dependency trees produced by trankit. Note that the y-axis is in log scale. In phrase structure terminology (e.g. on SLOG), *xcomp* recursion includes CP recursion and *nmod* recursion includes PP recursion.

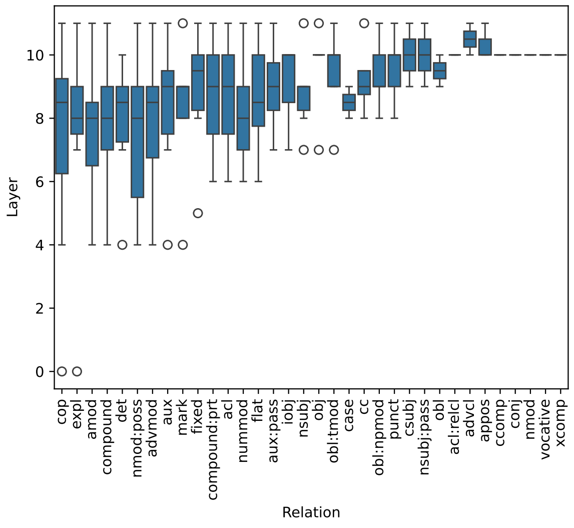

Figure B.3: Distribution of location of the look-up heads we identified per UD relation across the layers.

| **Generalization** | **STEP** | **Simple STEP** | **T5** | **T5+Dep Parse** |
|---|---|---|---|---|
| Deeper CP tail recursion | **74.5**±4.2 | 73.0±8.4 | 42.5±4.4 | 50.9±9.4 |
| Deeper PP recursion | **87.3**±2.5 | 78.5±4.4 | 75.0±4.5 | 70.8±5.9 |
| Deeper center embedding | 17.3±2.8 | **22.7**±2.2 | 8.9±0.4 | 11.5±2.1 |
| Shallower CP tail recursion | **100.0**±0.0 | **100.0**±0.0 | **100.0**±0.0 | **100.0**±0.0 |
| Shallower PP recursion | **100.0**±0.0 | **100.0**±0.0 | 99.9±0.2 | **100.0**±0.0 |
| Shallower center embedding | **100.0**±0.0 | **100.0**±0.0 | **100.0**±0.0 | **100.0**±0.0 |
| PP in indirect object NPs | 99.5±0.4 | 99.8±0.1 | **100.0**±0.0 | 99.7±0.1 |
| PP in subject NPs | **95.3**±0.1 | **95.3**±0.1 | 74.5±8.2 | **95.3**±0.2 |
| RC in indirect object NPs | 65.7±0.8 | 70.7±0.8 | **73.0**±0.8 | 67.1±1.2 |
| RC in subject NPs | **89.4**±0.9 | 88.4±1.4 | 70.4±2.6 | 69.1±12.3 |
| Indirect object-extracted RC | 16.5±13.0 | 11.7±13.9 | 62.9±12.3 | **73.8**±13.2 |
| Indirect object wh-questions | 78.8±18.3 | 77.8±14.6 | **100.0**±0.1 | 99.8±0.2 |
| Direct object wh-questions | 83.4±14.5 | 91.0±4.5 | **99.2**±0.6 | 83.8±18.0 |
| Wh-questions long movement | **55.8**±12.5 | 46.2±7.5 | 40.4±8.1 | 35.2±6.2 |
| Wh-questions with modified NPs | **85.9**±2.7 | 85.2±1.9 | 72.9±4.8 | 77.7±2.8 |
| Active subject wh-questions | **100.0**±0.1 | 99.1±1.8 | 99.6±0.3 | 97.1±5.6 |
| Passive subject wh-questions | 98.8±2.4 | 99.5±0.6 | **100.0**±0.0 | **100.0**±0.0 |
| Average | **79.3**±2.3 | 78.8±1.9 | 77.6±1.4 | 78.3±2.1 |

Table B.2: Full SLOG results.