Compositional Generalization Based on Semantic Interpretation: Where can Neural Networks Improve?

Stanford CS224N Custom Project

Carolyn Qu
Department of Symbolic Systems, Computer Science
Stanford University
cqu@stanford.edu

Rodrigo Nieto
Department of Computer Science
Stanford University
rjnieto@stanford.edu

Abstract

Recent research suggests that compositional generalization in Natural Language processing remains a challenge, even for state-of-the-art neural models such as Transformers. This paper explores various model configurations, techniques, and hyperparameters for the Transformer model to see how we can improve it on compositional generalization tasks. Within the design space of Transformers, we explore various techniques such as the decoding strategy, attention variants, oracle evaluation, and the optimizer type. We find that top-p sampling increases model accuracy across almost all the datasets we used to measure compositional generalization. This suggests that relying solely on greedy decoding can lead to suboptimal results, even in low-entropy tasks such as compositional generalization. Our research shows that because the greedy decoder is more prone to getting stuck in local minima, introducing entropy in the form of top-p sampling can actually be beneficial for the model. Our modified Transformer achieves state-of-the-art results in a semantic parsing compositional generalization benchmark (COGS), which we will publicly release so that one can reproduce our results. These findings could also have broader implications beyond compositional generalization testing, such as in the field of machine translation.

1 Key Information to include

- Mentor: John Hewitt
- External Collaborators (if you have any): None.
- Sharing project: No

2 Introduction

Compositional generalization is an important part of human language production; namely, the ability to understand and produce a potentially infinite number of novel, well-formed linguistic expressions by dynamically recombining known elements [1, 2, 3]. For instance, once a person is taught the meaning of the verb "dance," they can then understand the meaning of more complex phrases, such as "dance twice" or "sing and dance."

Recently, neural modeling approaches have taken over computational linguistics and Natural Language processing, using neural networks (designed to emulate human cognition) to tackle language processing tasks rather than handcrafted symbolic grammars [4]. Neural models greatly outperform grammar-based models in a diverse range of natural language tasks such as machine translation, question answering, and semantic parsing [4]. However, they fail rather dramatically on tests of compositional generalization, i.e., the ability to learn a set of basic primitives and combine them in more complex ways than those seen during training [5].

On the other hand, humans and classical AI algorithms (grammar and search-based systems) are not troubled by compositional tasks [6, 7]. This leaves the open question of whether we can build deep learning architectures that solve compositional tasks.

In this paper, we will focus on the Transformer architecture, which functions as the de facto standard for numerous NLP tasks. Despite recent efforts to improve the performance of Transformer models on compositional generalization, they often require advanced architecture changes or increased numbers of model parameters for promising results, leaving a lot to be desired [5, 8]. Our paper explores the design space of Transformers, while using a controlled number of model parameters and avoiding the use of large models which may lead to overfitting [5].

3 Related Work

The existing work surrounding compositional generalization includes a variety of approaches including data augmentation [9, 10, 11], Syntactic Attention [12], compositional parsers [6, 13], intermediate representations [14, 15] and structure annotations [7].

Additionally, previous work with Transforms suggests that by exploring the design space of Transformer models, minor architectural adjustments can drastically improve the performance of Transformers on compositional generalization tasks. Higher accuracy levels can be achieved without increasing the parameter count of the baseline model, but rather using relative positional encoding, changing hyperparameters, using a "copy decoder" and providing intermediate representations of data [8]. Similarly, other research suggests that revising model configurations such as the scaling of embeddings, early stopping, relative positional embedding, and Universal Transformer variants, can also drastically improve the performance of Transformers on compositional generalization. [5].

4 Approach

For consistency with our baseline, we will be building off the algorithm presented by Google Research [5] and revising the model architecture in order to see improvements in sequence level accuracy (check section 5.2 for definition). We specifically focused on areas of the Transformer that the modern literature has not extensively explored, such as decoding strategies, attention variants, and optimizers.

4.1 Top-p Sampling

At the current state of research, none of the state-of-the-art (SOTA) models presented deviate from using greedy decoding (see Figure 2a for an example). We would like to investigate this by implementing top-p sampling, which shortlists the tokens whose sum of likelihoods does not exceed a certain threshold value, p, which is defined as a hyperparameter [16]. The next token is then sampled from this shortlist based on likelihood scores. This top-p sampling method is commonplace for generating long-form text and high-entropy tasks [17, 18], but it has rarely been explored in the context of compositional generalization and other low-entropy tasks.

Although there is little precedent for using top-p sampling in lower-entropy tasks, one central problem with compositional generalization is the tendency for models to overfit to the dataset, leveraging learning shortcuts instead of learning compositional generalization. Top-p sampling tests how the model would perform once some degree of entropy is introduced.

This implementation adapts TensorFlow's code for two-dimensional predictions [19] to three-dimensions (batch size, sequence length, vocab size). At the current state, there is little open-source code for top-p sampling for three-dimensional predictions (as far as we are knowledgeable); therefore, even outside of testing compositional tasks, this implementation could be useful for other research.

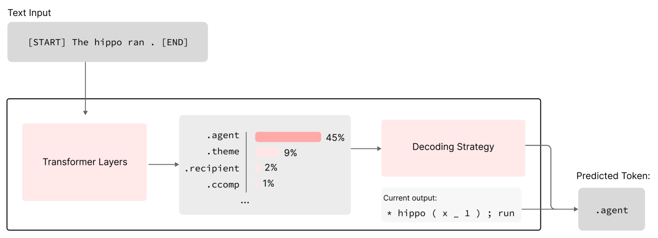


Figure 1: Example of encoder-decoder architecture for selecting next token [16].

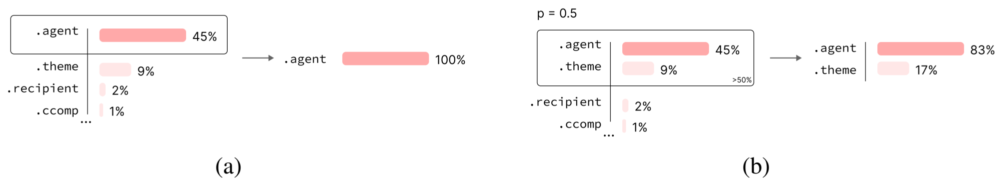


Figure 2: Greedy decoding vs top-p (nucleus) sampling. While greedy decoding selects the token with the highest probability, top-p sampling selects the top tokens whose likelihoods add up to *p* and samples based on the likelihood scores [16].

4.2 Attention Variants

Csordás et al. (2022) showed how the Transformer model drastically improved by using relative position encoding as opposed to absolute position encoding as they converge faster, are less sensitive to batch size, and help mitigate overfitting problems [8]. Therefore, the idea of relative attention, motivated us to try the effects of different attention variants. We thus experimented with the following attention variants: multiplicative attention (1), reduced-rank multiplicative attention (2), and additive attention (3).

$$e_i = s^\top W h_i \in \mathbb{R} \tag{1}$$

Where $e \in \mathbb{R}^N$, $h_1, ..., h_N \in \mathbb{R}^{d_1}$, $s \in \mathbb{R}^{d_2}$ and $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix.

$$e_i = s^\top (U^\top V) h_i = (Us)^\top (V h_i) \tag{2}$$

Where $U \in \mathbb{R}^{k \times k \times d_2}$, $V \in \mathbb{R}^{k \times k \times d_1}$ are low rank matrices, and $k << d_1, d_2$.

$$e_i = v^\top tanh(W_1 h_i + W_2 s) \in \mathbb{R} \tag{3}$$

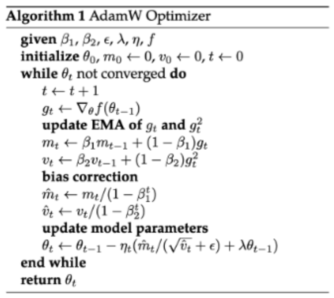
Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector, and d_3 is a hyperparameter.

4.3 Oracle Evaluation in Early Stopping

In Transformer models, the decision to end a decoding sequence is represented by the generation of a special end-of-string (EOS) token. Research shows that models often overfit to the position of the EOS token in the train set, suggesting that Transformers struggle to generalize to longer output lengths than they are trained for [20]. *Oracle evaluation* measures whether the models are otherwise able to solve the compositionality task, this is done by ignoring the EOS token during evaluation and using the ground-truth sequence length to stop decoding.

4.4 Lion Optimizer

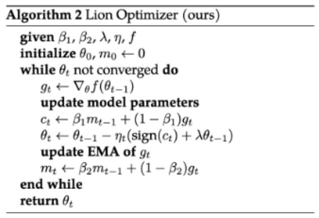
A very recent paper by Chen et al. (2023) presented a simple and effective optimization algorithm Lion that is more memory-efficient and has better in-context learning ability – an emergent behavior in large language models where the LLM performs a task just by conditioning on input-output examples, without optimizing any parameters [21] – than Adam [22]. The paper proposes discovering



Algorithm 1 AdamW Optimizer

```
given $\beta_1, \beta_2, \epsilon, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    update EMA of $g_t$ and $g_t^2$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    bias correction
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
    update model parameters
    $\theta_t \leftarrow \theta_{t-1} - \eta_t(\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda\theta_{t-1})$
end while
return $\theta_t$
```

(a)



Algorithm 2 Lion Optimizer (ours)

```
given $\beta_1, \beta_2, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0$
while $\theta_t$ not converged do
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    update model parameters
    $c_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $\theta_t \leftarrow \theta_{t-1} - \eta_t(\text{sign}(c_t) + \lambda\theta_{t-1})$
    update EMA of $g_t$
    $m_t \leftarrow \beta_2 m_{t-1} + (1 - \beta_2) g_t$
end while
return $\theta_t$
```

(b)

Figure 3: Adam (a) and Lion (b) optimizer algorithms [22]

optimization algorithms through the usage of program search. Their method was able to discover the Lion optimizer. Thus, we were curious to see the performance of Lion relative to Adam, given its improvement in some NLP applications. The differences between the Lion Optimizer algorithm and the AdamW algorithm are highlighted in Figure 3.

5 Experiments

5.1 Data

Our primary evaluation dataset will be **COGS** (Compositional Generalization Challenge based on Semantic Interpretation): a semantic parsing dataset based on a fragment of English [23], which tests the task of semantic parsing to map an English sentence to a logical form. For example, an input token: `[START] The hippo ran . [END]` is mapped to the logical form: `* hippo ( x _ 1 ) ; run . agent ( x _ 2 , x _ 1 ) [END] .`

Training on compositional task-based datasets often leads to overfitting[24], namely due to uncontrolled lexical exposure during training. Therefore, it is imperative that we test our models against a suite of different compositional generalization tasks. The datasets are as follows:

- **Algorithmic Datasets**: These are datasets that evaluate *productivity*-style compositional generalization: extrapolation to longer sequences than those seen during training [5]. This set includes addition (*Add*), addition of negatives (*AddNeg*), reversing an input sequence (*Reverse*), duplication of a sequence (*Dup*), Cartesian product of two sequences (*Cart*), and the intersection of two sequences (*Inters*).
- **SCAN** (Simplified versions of the CommAI Navigation tasks): A dataset for grounded navigation which consists of a set of simple compositional navigation commands paired with the corresponding action sequences [25]. We include the length split *SCAN-l*, and the add primitive jump split *SCAN-aj*.
- **CFQ** (Compositional Freebase Questions): A large and realistic natural language question answering dataset [26]. We include the MCD1 split *CFQ*. This dataset requires preprocessing to convert tfrecords to plain text.
- **PCFG** (Probabilistic Context Free Grammar String Edit Task): A dataset with sequence-to-sequence problems specifically designed to test different systematicity, productivity, substitutivity, localism, and overgeneralization [27]. We include the productivity *PCFG-p* and systematicity *PCFG-s* splits of the dataset.

For each of the datasets, we focus on a generalization split, where the test set is a sample from a distribution that is systematically different from the one for training.

5.2 Evaluation method

For consistency with our initial baseline, we will evaluate the compositional generalization of our modified Transformer on the sequence level accuracy of the output. Sequence level accuracy, in this context, is defined as the percentage of entire sequences that are correctly predicted by the model within a dataset. Therefore, in order to get 100% sequence accuracy, the token output prediction must token-by-token match the ground truth in that precise order.

5.3 Experimental details

- **Transformer Variant:** We will be implementing top-p on our baseline model provided by Ontanón et al. (2021)[5]. This will involve *relative position encoding*, where the relative position labels define a learnable embedding that is added to the key during the attention process. However, besides that we will not use *weight sharing*, unlike the work in the paper considering the Csordás et al. (2022) research notes that weight sharing actually limits the capacity of the Transformer model [8]. Finally, we will also avoid using the *copy decoder* as used by Ontanón et al. (2021). The reason behind this decision is that the inclusion of a copy decoder would drastically increase the complexity of the model – which we will explain why in the **Model Configurations** section is not in the scope of our project – and copy mechanisms can often lead to repetition or redundancy in the output, which might lead to overfitting to certain n-grams.
- **Model Configurations:** The hypothesis from the Ontanón et al. (2021) investigation concludes with insight that most large models greatly outperform their respective small ones because large models tend to overfit, which is to be expected [5]. Therefore, for our model configurations, we wanted to small model size with the number of encoder / decoder layers set to 2, the dimensionality of the token embeddings set to 64, the intermediate dimensionality used by the feed-forward sublayer set to 256, and the number of attention-heads in the attention sublayers set to 4.
- **Top-p Sampling:** For top-p sampling, we must set a parameter for the p value to limit the long tail of low-probability tokens that may be sampled [16]. In this practice, we found that the p value that worked most effectively was when p was set to 0.9.
- **Batch Size, Learning Rate, and Optimizer Betas:** In order to keep consistency with our baseline, we will be utilizing the same batch size of 64 and a custom schedule learning rate with the number of warmup steps set to 400. Similarly, for the Adam optimizer, we also used a β_1 of 0.9 and a β_2 of 0.98, as well as ϵ of $1e-9$. Furthermore, we found the same Adam optimizer parameters were the best-performing hyperparameters on Lion as well.
- **Loss function:** As used in the Ontanón et al. (2021) work, we will be using the cross entropy loss function, which as shown in Csordás et al. (2022) tends to be the standard for compositional generalization tasks [8].

5.4 Results

We report our sequence level accuracy across all datasets in Table 1. All results that are reported are run chronologically and are meant to find the best combination of techniques to optimize for accuracy, meaning that because we noticed that top-p sampling greatly improved overall accuracy, all the other experiments (i.e. Multiplicative Attention, Reduced-rank multiplicative Attention, Additive attention, Oracle evaluation, and the Lion optimizer) included top-p sampling.

	Add	AddNeg	Reverse	Dup	Cart	Inters	Scan-l	Scan-aj	PCFG-p	PCFG-s	COGS	CFQ	Avg
Baseline	0.004	0.018	0.422	0.486	0.004	0.501	0.064	0.003	0.238	0.451	0.170	0.322	0.224
Top-p	0.004	0.021	0.425	0.793	0.000	0.500	0.050	0.012	0.201	0.514	0.296	0.355	0.264
Multi Att	0.002	0.087	0.024	0.002	0.000	0.507	0.091	0.027	0.198	0.514	0.103	0.312	0.156
Reduced Att	0.000	0.001	0.002	0.000	0.000	0.511	0.021	0.001	0.028	0.022	0.047	0.183	0.068
Add Att	0.000	0.001	0.000	0.000	0.000	0.500	0.018	0.001	0.022	0.024	0.023	0.150	0.062
Oracle	0.013	0.010	0.392	0.439	0.000	0.500	0.103	0.036	0.309	0.467	0.304	0.296	0.239
Lion	0.000	0.002	0.394	0.142	0.000	0.502	0.073	0.021	0.114	0.256	0.078	0.232	0.151

Table 1: Comparison of the sequence level accuracy for various implementations, across all datasets with a "rel-e" size model compared to our baseline with all other hyperparameters kept the same [5]. All results after top-p include top-p sampling in their implementation to improve on the previous run.

Our results show that top-p sampling significantly increased compositional generalization scores, increasing the average sequence level accuracy from the baseline of **22.4%** to **26.4%**. The improvements for top-p sampling were especially significant in *Dup, PCFG-s,* and *COGS*. This was quite surprising and unexpected, namely because top-p sampling is often used for high-entropy tasks. We further analyze the top-p sampling in Section 6 to explore why introducing entropy might be beneficial for compositional generalization tasks.

Improvements from attention variants were inconsistent. Multi-headed attention significantly increased performance on *AddNeg, SCAN-l, SCAN-aj*, but worse overall. Reduced-Rank attention increased scores on *Inters* and *COGS* but significantly decreased performance on other datasets. Additive attention performed worse than the baseline across the board. Despite relative position encoding overall helping compositional generalization, introducing attention weights appears to hurt the sequence level accuracy of the model. Additionally, increasing the number of weight matrices for reduced-rank and additive attention seemed to further weaken scores.

Oracle evaluation seemed to perform variably, increasing scores on *Add, SCAN-l, SCAN-aj, PCFG, PCFG-p, and COGS*. However, top-p sampling with Oracle Evaluation seemed to perform worse overall than top-p sampling on its own.

The Lion Optimizer also performed rather poorly. This behavior was unexpected, namely because in comparison to Adam, Lion significantly performs better on several NLP applications [22]. It is unclear where Lion underperforms compared to Adam, even with the use of our best-performing hyperparameters. We assume this is partly due to Lion being a new algorithm, with limited research or understanding of its behavior and properties at the time of writing.

6 Analysis

From our results, we focused on exploring how the introduction of entropy through top-p sampling can enhance compositional generalization. Our initial results showed significant improvement across several datasets, but we decided to examine the effects of top-p sampling on COGS in greater detail. To this end, we conducted a comparative analysis between the outputs generated by top-p sampling and our baseline, specifically looking for areas of divergence.

6.1 Top-p sampling: Probability Distribution

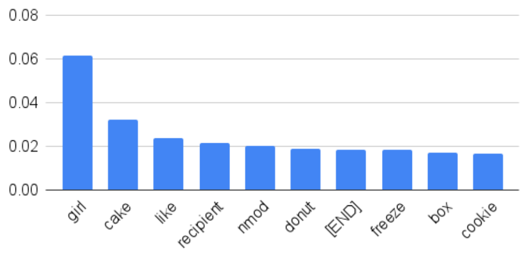


Figure 4: An example probability distribution for the first token in COGS

Our analysis revealed that when COGS produced an output for the generalization set, the initial words had a low likelihood of occurrence (ranging from 5% to 14%), while the later words had much higher likelihood scores (up to 99%). This finding highlights the role that top-p sampling plays in generating our outputs, and provides us insight on its impact on compositional generalization.

Additionally, the $*$ symbol was produced more frequently in the baseline prediction as compared to top-p. Given the low confidence on the first few tokens, we hypothesize that the baseline model learns a shortcut to produce $*$ as the first token. In the top-p variant of our baseline, the model is forced to consider other alternatives as the first token, which is especially useful as the likelihood for the first token in COGS is rather low.

6.2 Sequence level Accuracy In COGS

Both our baseline model and our top-p model performed variably depending on the dataset (For more detailed information see A.1). We noticed that top-p performed significantly better on the

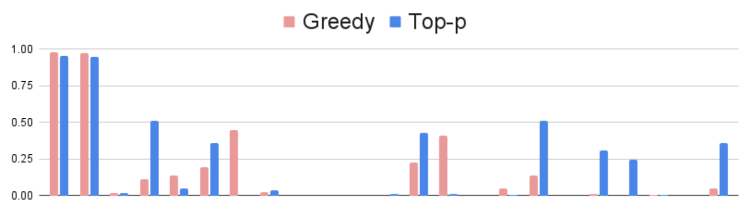


Figure 5: Sequence level accuracies for the 22 generalization splits evaluated in COGS

following data splits: *Subject → Object (proper noun), Object → Subject (proper noun), Object-omitted transitive → Transitive, PP dative → Double object dative, Theme NP → Obiect-omitted transitive Subiect, Depth generalization: Sentential complements*.

We hypothesize that the semantic structure in these data splits is easier to learn. The similar increase in performance for the *Subject → Object (proper noun)*, *Object → Subject (proper noun)* data splits (and lack of increase in their *common noun* counterpart datasets) suggests that some form of generalization occurs across the terms and the structure learned in these splits.

6.3 Common Resolved Error Types

In analyzing some of the common types of sequences that top-p predicted correctly and greedy decoding predicted incorrectly, we noticed that the overwhelming majority of the greedy-decoded outputs were only off by one token.

- **1st Token Errors:** One of the most frequent sequence types we observed was a *1st-token error*, where the greedy decoder incorrectly the first token of the sequence, which top-p sampling correctly calculated.

```
Input:   [START] A cockroach expected to nap .  [END]
Target:  cockroach(x_1)AND expect.agent(x_2,x_1)AND expect.xcomp(x_2,x_4)AND nap.agent(x_4,x_1)[END]
Top-p :  cockroach(x_1)AND expect.agent(x_2,x_1)AND expect.xcomp(x_2,x_4)AND nap.agent(x_4,x_1) [END]
Greedy:  cake(x_1)AND expect.agent(x_2,x_1)AND expect.xcomp(x_2,x_4)AND nap.agent(x_4,x_1)[END]
```

- **Definite article errors:** In the *Subject → Object, proper noun* dataset, 85% of the sequences that were mispredicted by the Greedy Decoder and accurately predicted by top-p sampling included the definite article *the* A.2. On the other hand, both Greedy decoding and top-p sampling incorrectly predicted many of the sequences that included an indefinite article.

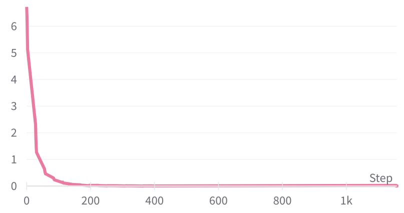
```
Input:   [START] The hippo dusted .  [END]
Target:  *hippo(x_1);dust.agent(x_2,x_1)[END][END]
Top-p :  *LAMBDA(x_1);dust.agent(x_2,x_1)[END][END]
Greedy:  *hippo(x_1);dust.agent(x_2,x_1)[END][END]
```

By analyzing various error types across datasets, we can observe that greedy decoding fails on instances of test data that it hasn't seen before, it predicts a word *similar* to the input word. For instance, if the input includes the word `cockroach`, the model may often output the word `cake` in the prediction in place of the correct token `cockroach`, resulting in a low sequence level accuracy ($\sim 0\%$, especially if the word `cockroach` appears in multiple examples in the dataset.

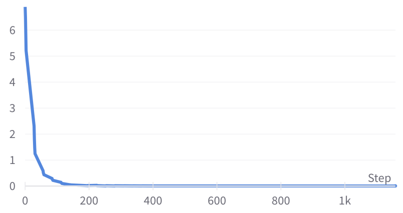
6.4 Entropy in Top-p

Although top-p sampling is not infallible, it allows for the model to consider other alternatives. On both top-p and greedy decoding models, the loss rapidly approaches 0 (typically < 1 by 25% through training, as shown in Figure 6) and thus backpropagation may not have a significant enough effect later in training because the error signal propagated through the network is relatively small.

This could cause the weights and the biases to update less drastically, making it difficult for the model to escape a local minimum. Even with the entropy introduced through the cross-entropy loss function, the greedy decoding model still struggles to push the optimization algorithm out of a local minimum. In this case, the model could get stuck at `cake` instead of finding the global optimum, `cockroach`.



(a)



(b)

Figure 6: Training loss for greedy (a) and top-p (b) models

We hypothesize that top-p sampling mitigates this issue; if the token with the second highest likelihood is `cockroach` instead of `cake`, there is a higher probability that the token `cockroach` would be chosen, thus helping the model to escape the local minimum created around `cake`. If the overwhelming majority of the greedy-decoded outputs were only off by one token, this would put the sequence level accuracy to 0% for all those predictions. This is exacerbated by the fact that the sequence level accuracy will not achieve much learning through backpropagation if the value of the loss function is low, thus, showing the need to deviate from pure greedy decoding and introduce entropy through other decoding strategies.

7 Conclusion

In this work, we tested various model configurations and techniques such as decoding strategy, attention variants, oracle evaluation, and optimizer type, and found that top-p sampling produced significant increases in accuracy across all compositional generalization datasets. We find that top-p sampling has promising potential for low-entropy tasks such as compositional generalization, and our modified Transformer model results in a 12.6% gain in COGS compared to the current SOTA [5]. From our qualitative analysis, we found that by increasing entropy, top-p sampling helps Transformer models escape local minima and accurately predict tokens that hadn't been seen before in training.

One major limitation of our work is the initial model configurations; due to GPU and time limitations, we were only able to base our model on the smallest version of our baseline [5]. For future work, we would like to further explore our hypotheses about how top-p sampling helps a Transformer model escape local minima, as well as further algorithm modifications, i.e. dynamically altering the threshold value p across multiple datasets.

Additionally, we would like to explore the merits of our findings on other low-entropy tasks such as machine translation, which have little precedent in utilizing decoding strategies other than greedy decoding. Compositional generalization is particularly useful for machine translation models, as it allows models to handle more complex sentences and generate translations for sentences that haven't been seen before. This would be especially meaningful for languages that lack enough data and resources to make a robust and accurate translation model. Poor compositional generalization leads to inaccurate and nonsensical translations, which is commonplace in endangered and low-density languages that lack robust datasets.

In conclusion, there is a need for further exploration of the role of entropy in machine translation models or low-entropy tasks such as compositional generalization and machine translation. By gaining a deeper understanding of how entropy can be incorporated into learning algorithms, we may be able to enhance the performance of machine translation systems.

References

[1] Noam Chomsky. Logical structure in language. *Journal of the American Society for Information Science*, 8(4):284, 1957.

[2] Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[3] Jerry A Fodor and Ernest Lepore. *The compositionality papers*. Oxford University Press, 2002.

[4] Lucia Donatelli and Alexander Koller. Compositionality in computational linguistics. *Annual Review of Linguistics*, 9, 2023.

[5] Santiago Ontanón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. Making transformers solve compositional tasks. *arXiv preprint arXiv:2108.04378*, 2021.

[6] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*, 2020.

[7] Juyong Kim, Pradeep Ravikumar, Joshua Ainslie, and Santiago Ontañón. Improving compositional generalization in classification tasks via structure annotations. *arXiv preprint arXiv:2106.10434*, 2021.

[8] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The devil is in the detail: Simple tricks improve systematic generalization of transformers. *arXiv preprint arXiv:2108.12284*, 2021.

[9] Jacob Andreas. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545*, 2019.

[10] Yichen Jiang, Xiang Zhou, and Mohit Bansal. Mutual exclusivity training and primitive augmentation to induce compositionality. *arXiv preprint arXiv:2211.15578*, 2022.

[11] Arkil Patel, Satwik Bhattamishra, Phil Blunsom, and Navin Goyal. Revisiting the compositional generalization abilities of neural sequence models. *arXiv preprint arXiv:2203.07402*, 2022.

[12] Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*, 2019.

[13] Pia Weißenhorn, Yuekun Yao, Lucia Donatelli, and Alexander Koller. Compositional generalization requires compositional parsers. *arXiv preprint arXiv:2202.11937*, 2022.

[14] Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. Unlocking compositional generalization in pre-trained models using intermediate representations. *arXiv preprint arXiv:2104.07478*, 2021.

[15] Soham Dan, Osbert Bastani, and Dan Roth. Understanding robust generalization in learning regular languages. In *International Conference on Machine Learning*, pages 4630–4643. PMLR, 2022.

[16] Top-k & top-p. *co:here*, 2021. `https://docs.cohere.ai/docs/controlling-generation-with-top-k-top-p`.

[17] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[18] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751v2*, 2020.

[19] Tensorflow. sampling_module. 2022. `https://github.com/tensorflow/models/blob/master/official/nlp/modeling/ops/sampling_module.py`.

[20] Benjamin Newman, John Hewitt, Percy Liang, and Christopher D Manning. The eos decision and length extrapolation. *arXiv preprint arXiv:2010.07174*, 2020.

[21] Sang Michael Xie and Sewon Min. How does in-context learning work? a framework for understanding the differences from traditional supervised learning. *The Stanford AI Lab Blog*, 2022.

[22] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675v2*, 2023.

[23] Najoung Kim and Tal Linzen. Cogs: A compositional generalization challenge based on semantic interpretation. *arXiv preprint arXiv:2010.05465*, 2020.

[24] Najoung Kim, Tal Linzen, and Paul Smolensky. Uncontrolled lexical exposure leads to overestimation of compositional generalization in pretrained models. *arXiv preprint arXiv:2212.10769*, 2022.

[25] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR, 2018.

[26] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. Measuring compositional generalization: A comprehensive method on realistic data. *arXiv preprint arXiv:1912.09713*, 2019.

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

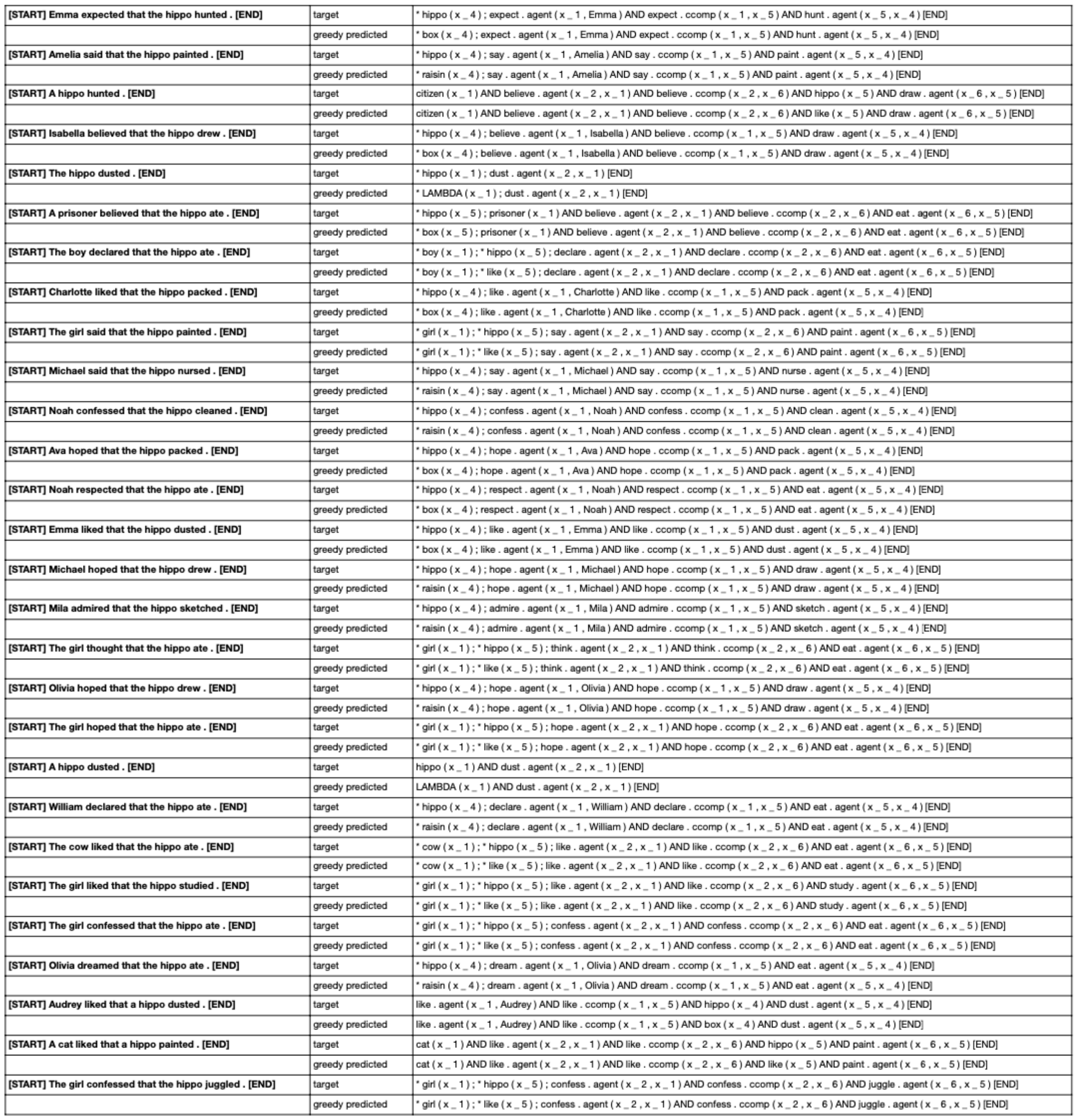
A Appendix (optional)

A.1 COGS dataset splits

Encoding Type	Greedy	Top-p
Test split	0.9782	0.9507
Dev split	0.9728	0.9442
Subject → Object (common noun)	0.0177	0.0145
Subject → Object (proper noun)	0.1145	0.5093
Object → Subject (common noun)	0.1343	0.0479
Object → Subject (proper noun)	0.1968	0.3572
Primitive noun → Subject (common noun)	0.4447	0.001
Primitive noun → Subject (proper noun)	0.026	0.0375
Primitive noun → Object (common noun)	0.00	0.00
Primitive noun → Object (proper noun)	0.00	0.00
Primitive verb → Infinitival argument	0.00	0.00
Active → Passive	0.001	0.0135
Passive → Active	0.226	0.4281
Object-omitted transitive → Transitive	0.4062	0.0114
Unaccusative → Transitive	0.00	0.00
Double obiect dative → PP dative	0.05	0.0031
PP dative → Double obiect dative	0.1395	0.5083
Agent NP → Unaccusative Subiect	0.001	0.00
Theme NP → Obiect-omitted transitive Subiect	0.0083	0.3062
Theme NP → Unergative subject	0.00	0.2447
Object-modifying PP →Subject-modifying PP	0.0031	0.0052
Depth generalization: PP modifiers	0.00	0.00
Depth generalization: Sentential complements	0.051	0.3572

Table 2: Sequence level accuracy in different subsets of the generalization set in COGS for both greedy-decoding (baseline) and top-p [5]

A.2 Recovered Errors for dataset 3 (*Subject → Object, proper noun*)



[START] Emma expected that the hippo hunted . [END]	target	* hippo (x _ 4) ; expect . agent (x _ 1 , Emma) AND expect . ccomp (x _ 1 , x _ 5) AND hunt . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; expect . agent (x _ 1 , Emma) AND expect . ccomp (x _ 1 , x _ 5) AND hunt . agent (x _ 5 , x _ 4) [END]
[START] Amelia said that the hippo painted . [END]	target	* hippo (x _ 4) ; say . agent (x _ 1 , Amelia) AND say . ccomp (x _ 1 , x _ 5) AND paint . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; say . agent (x _ 1 , Amelia) AND say . ccomp (x _ 1 , x _ 5) AND paint . agent (x _ 5 , x _ 4) [END]
[START] A hippo hunted . [END]	target	citizen (x _ 1) AND believe . agent (x _ 2 , x _ 1) AND believe . ccomp (x _ 2 , x _ 6) AND hippo (x _ 5) AND draw . agent (x _ 6 , x _ 5) [END]
	greedy predicted	citizen (x _ 1) AND believe . agent (x _ 2 , x _ 1) AND believe . ccomp (x _ 2 , x _ 6) AND like (x _ 5) AND draw . agent (x _ 6 , x _ 5) [END]
[START] Isabella believed that the hippo drew . [END]	target	* hippo (x _ 4) ; believe . agent (x _ 1 , Isabella) AND believe . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; believe . agent (x _ 1 , Isabella) AND believe . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
[START] The hippo dusted . [END]	target	* hippo (x _ 1) ; dust . agent (x _ 2 , x _ 1) [END]
	greedy predicted	* LAMBDA (x _ 1) ; dust . agent (x _ 2 , x _ 1) [END]
[START] A prisoner believed that the hippo ate . [END]	target	* hippo (x _ 5) ; prisoner (x _ 1) AND believe . agent (x _ 2 , x _ 1) AND believe . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* box (x _ 5) ; prisoner (x _ 1) AND believe . agent (x _ 2 , x _ 1) AND believe . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] The boy declared that the hippo ate . [END]	target	* boy (x _ 1) ; * hippo (x _ 5) ; declare . agent (x _ 2 , x _ 1) AND declare . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* boy (x _ 1) ; * like (x _ 5) ; declare . agent (x _ 2 , x _ 1) AND declare . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] Charlotte liked that the hippo packed . [END]	target	* hippo (x _ 4) ; like . agent (x _ 1 , Charlotte) AND like . ccomp (x _ 1 , x _ 5) AND pack . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; like . agent (x _ 1 , Charlotte) AND like . ccomp (x _ 1 , x _ 5) AND pack . agent (x _ 5 , x _ 4) [END]
[START] The girl said that the hippo painted . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; say . agent (x _ 2 , x _ 1) AND say . ccomp (x _ 2 , x _ 6) AND paint . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; say . agent (x _ 2 , x _ 1) AND say . ccomp (x _ 2 , x _ 6) AND paint . agent (x _ 6 , x _ 5) [END]
[START] Michael said that the hippo nursed . [END]	target	* hippo (x _ 4) ; say . agent (x _ 1 , Michael) AND say . ccomp (x _ 1 , x _ 5) AND nurse . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; say . agent (x _ 1 , Michael) AND say . ccomp (x _ 1 , x _ 5) AND nurse . agent (x _ 5 , x _ 4) [END]
[START] Noah confessed that the hippo cleaned . [END]	target	* hippo (x _ 4) ; confess . agent (x _ 1 , Noah) AND confess . ccomp (x _ 1 , x _ 5) AND clean . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; confess . agent (x _ 1 , Noah) AND confess . ccomp (x _ 1 , x _ 5) AND clean . agent (x _ 5 , x _ 4) [END]
[START] Ava hoped that the hippo packed . [END]	target	* hippo (x _ 4) ; hope . agent (x _ 1 , Ava) AND hope . ccomp (x _ 1 , x _ 5) AND pack . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; hope . agent (x _ 1 , Ava) AND hope . ccomp (x _ 1 , x _ 5) AND pack . agent (x _ 5 , x _ 4) [END]
[START] Noah respected that the hippo ate . [END]	target	* hippo (x _ 4) ; respect . agent (x _ 1 , Noah) AND respect . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; respect . agent (x _ 1 , Noah) AND respect . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
[START] Emma liked that the hippo dusted . [END]	target	* hippo (x _ 4) ; like . agent (x _ 1 , Emma) AND like . ccomp (x _ 1 , x _ 5) AND dust . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* box (x _ 4) ; like . agent (x _ 1 , Emma) AND like . ccomp (x _ 1 , x _ 5) AND dust . agent (x _ 5 , x _ 4) [END]
[START] Michael hoped that the hippo drew . [END]	target	* hippo (x _ 4) ; hope . agent (x _ 1 , Michael) AND hope . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; hope . agent (x _ 1 , Michael) AND hope . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
[START] Mila admired that the hippo sketched . [END]	target	* hippo (x _ 4) ; admire . agent (x _ 1 , Mila) AND admire . ccomp (x _ 1 , x _ 5) AND sketch . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; admire . agent (x _ 1 , Mila) AND admire . ccomp (x _ 1 , x _ 5) AND sketch . agent (x _ 5 , x _ 4) [END]
[START] The girl thought that the hippo ate . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; think . agent (x _ 2 , x _ 1) AND think . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; think . agent (x _ 2 , x _ 1) AND think . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] Olivia hoped that the hippo drew . [END]	target	* hippo (x _ 4) ; hope . agent (x _ 1 , Olivia) AND hope . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; hope . agent (x _ 1 , Olivia) AND hope . ccomp (x _ 1 , x _ 5) AND draw . agent (x _ 5 , x _ 4) [END]
[START] The girl hoped that the hippo ate . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; hope . agent (x _ 2 , x _ 1) AND hope . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; hope . agent (x _ 2 , x _ 1) AND hope . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] A hippo dusted . [END]	target	hippo (x _ 1) AND dust . agent (x _ 2 , x _ 1) [END]
	greedy predicted	LAMBDA (x _ 1) AND dust . agent (x _ 2 , x _ 1) [END]
[START] William declared that the hippo ate . [END]	target	* hippo (x _ 4) ; declare . agent (x _ 1 , William) AND declare . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; declare . agent (x _ 1 , William) AND declare . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
[START] The cow liked that the hippo ate . [END]	target	* cow (x _ 1) ; * hippo (x _ 5) ; like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* cow (x _ 1) ; * like (x _ 5) ; like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] The girl liked that the hippo studied . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND study . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND study . agent (x _ 6 , x _ 5) [END]
[START] The girl confessed that the hippo ate . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; confess . agent (x _ 2 , x _ 1) AND confess . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; confess . agent (x _ 2 , x _ 1) AND confess . ccomp (x _ 2 , x _ 6) AND eat . agent (x _ 6 , x _ 5) [END]
[START] Olivia dreamed that the hippo ate . [END]	target	* hippo (x _ 4) ; dream . agent (x _ 1 , Olivia) AND dream . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
	greedy predicted	* raisin (x _ 4) ; dream . agent (x _ 1 , Olivia) AND dream . ccomp (x _ 1 , x _ 5) AND eat . agent (x _ 5 , x _ 4) [END]
[START] Audrey liked that a hippo dusted . [END]	target	like . agent (x _ 1 , Audrey) AND like . ccomp (x _ 1 , x _ 5) AND hippo (x _ 4) AND dust . agent (x _ 5 , x _ 4) [END]
	greedy predicted	like . agent (x _ 1 , Audrey) AND like . ccomp (x _ 1 , x _ 5) AND box (x _ 4) AND dust . agent (x _ 5 , x _ 4) [END]
[START] A cat liked that a hippo painted . [END]	target	cat (x _ 1) AND like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND hippo (x _ 5) AND paint . agent (x _ 6 , x _ 5) [END]
	greedy predicted	cat (x _ 1) AND like . agent (x _ 2 , x _ 1) AND like . ccomp (x _ 2 , x _ 6) AND like (x _ 5) AND paint . agent (x _ 6 , x _ 5) [END]
[START] The girl confessed that the hippo juggled . [END]	target	* girl (x _ 1) ; * hippo (x _ 5) ; confess . agent (x _ 2 , x _ 1) AND confess . ccomp (x _ 2 , x _ 6) AND juggle . agent (x _ 6 , x _ 5) [END]
	greedy predicted	* girl (x _ 1) ; * like (x _ 5) ; confess . agent (x _ 2 , x _ 1) AND confess . ccomp (x _ 2 , x _ 6) AND juggle . agent (x _ 6 , x _ 5) [END]