Improving Compositional Generalization in Classification Tasks via Structure Annotations

Juyong Kim **Pradeep Ravikumar**
Carnegie Mellon University
{juyongk,pradeepr}@cs.cmu.edu

Joshua Ainslie **Santiago Ontañón**
Google Research
{jainslie,santiontanon}@google.com

Abstract

Compositional generalization is the ability to generalize systematically to a new data distribution by combining known components. Although humans seem to have a great ability to generalize compositionally, state-of-the-art neural models struggle to do so. In this work, we study compositional generalization in classification tasks and present two main contributions. First, we study ways to convert a natural language sequence-to-sequence dataset to a classification dataset that also requires compositional generalization. Second, we show that providing structural hints (specifically, providing parse trees and entity links as attention masks for a Transformer model) helps compositional generalization.

1 Introduction

Compositional generalization is the ability of a system to systematically generalize to a new data distribution by combining known components or primitives. For example, assume a system has learned the meaning of “jump” and that “jump twice” means that the action “jump” has to be repeated two times. Upon learning the meaning of the action “jax”, it should be able to infer what “jax twice” means. Although modern neural architectures are pushing the state of the art in many complex natural language tasks, these models still struggle with compositional generalization (Hupkes et al., 2020).

In order to advance research in this important direction, in this paper we present two main contributions [1]. First, we present a binary classification dataset which is hard in a compositional way. This allows for studying the compositional generalization ability of a larger range of models than sequence generation tasks, since the task only requires an encoder, and not a decoder. Specifically, we present a methodology to convert an existing semantic parsing dataset, CFQ (Keysers et al., 2019), into a binary classification dataset that is also compositionally hard.

Our second and main contribution is showing that a transformer-based model can better generalize compositionally if we provide hints on the structure of the input. Specifically, we do so by modifying the attention mask used by the model. This is an interesting result, as (except for two additions, which we elaborate on in Section 4) attention masks do not “add” any attention capabilities to the model. Instead, it seems that it is the removal of certain attention pairs that makes the difference. This suggests that vanilla Transformer is having a hard time suppressing non-compositional attention.

2 Background

This section overviews existing work on compositional generalization and then some background on the Transformer models used in this paper. Please see Section B in the appendix for detailed review.

Compositional Generalization. Compositional generalization can manifest in different ways (Hupkes et al., 2020) such as *systematicity* (recombination of known parts and rules) or *productivity* (extrapolation to longer sequences than those seen during training), among others. Early work focused on showing how different deep learning models do not generalize compositionally (Liška et al., 2018), and datasets such as SCAN (Lake and Baroni, 2018) or CFQ (Keysers et al., 2019) were proposed to show these effects.

Work toward improving compositional generalization has proposed ideas such as Syntactic attention (Russin et al., 2019), increased pre-training (Furrer et al., 2020), data augmentation (Andreas, 2019), or general purpose sequential models such as *Neural Turing Machines* or *Differ-*

[1] http://goo.gle/compositional-classification

ential Neural Computers (Graves et al., 2016).

ETC. For our experimental evaluation we use the ETC (Ainslie et al., 2020) Transformer model. ETC extends the standard Transformer model in 3 key ways: (1) it uses a global-local attention mechanism to scale to long inputs, (2) it uses relative attention (Shaw et al., 2018) and flexible masking and (3) it uses a new pre-training loss based on Contrastive Predictive Coding (CPC) (Oord et al., 2018). The last two extensions allow it to handle structured inputs containing, for example, hierarchical structure. In this work, we rely on (2) to annotate the structure of the input.

3 The CFQ Classification Dataset

The Compositional Freebase Questions (CFQ) dataset (Keysers et al., 2019) is an NLU dataset to measure the compositional capability of a learner. It is designed around the task of *translating* a natural language question into a SPARQL query. The dataset has been automatically generated by a grammar and contains 239,357 sentence/query pairs. An example is shown in Figure 3a.

As shown in the original work of Keysers et al. (2019) in order to properly measure the compositional generalization ability of a model, the train and test sets should be split with similar distributions of tokens (*atoms*), but different distributions of their compositions (the *compounds*). In the CFQ dataset, to ensure this, two divergences, namely *atom divergence* and *compound divergence*, between the train and dev/test set are measured while constructing the splits. As a result, carefully selected splits called *maximum compound divergence (MCD)* splits are hard for standard neural networks (they perform well in the train set, but poorly in the test set), while the random splits are easier.

We convert the CFQ dataset into a dataset with a binary classification task. In this new dataset, the input is a question and a SPARQL query, and the task is to determine whether these two sequences have the same meaning or not. Two considerations must be made to ensure the resulting dataset requires compositional generalization:

Negative Example Strategies: Positive instances of the binary classification task can be obtained directly from the original dataset, but to obtain negatives, we use either of two strategies:

- Random negatives: We pair each question with a randomly chosen query.
- Model negatives: Using baseline models (LSTM (Hochreiter and Schmidhuber, 1997), Transformer (Vaswani et al., 2017), and Universal Transformer (Dehghani et al., 2018)) trained on the original CFQ dataset, we get top-k query predictions for each question. After filtering syntactically invalid queries and duplicates, we can get hard examples for classification from their incorrect predictions.

Model negatives are important, as otherwise, the task becomes too easy and would likely not require compositional generalization. See Figure 1 for examples of random/model negative instances.

Compound Distribution of Negative Examples: To prevent data leakage (e.g., compounds from the test set of the original CFQ dataset leaking into the training set of the classification CFQ dataset), we carefully choose the sampling set for random negatives and the train and inference set for model negatives. We generate two splits of the original CFQ dataset. Each split contains three sets with 50% data on train, 25% on dev and 25% on test. The first is a *random split* of the data, and the second (*MCD split*), maximizes the compound divergence between train and dev/test using the same method as in the original CFQ work. Then, we process the examples in each of these sets generating positive and negative examples. For random negatives, we sample negative queries for each questions from the set which the original example belongs to (train/dev/test). For model negatives, to generate negatives for the training set, we divide it into two halves, train models in one, and generate negatives with the other half. For dev/test, we train on dev and generate negatives on test, and vice versa. Figure 2 illustrates this procedure, designed to ensure there is no leakage of compounds between train and dev/test.

For both strategies, we make 1 positive and 3 negatives per original CFQ example. Also, we set aside 5% of the train set as a hold-out set to check i.i.d. generalization.

4 Compositional Generalization via Structure Annotation

Our hypothesis is that part of the difficulty in compositional generalization is to parse the structure of the input. To test this, we evaluate the performance of models when we provide annotations for two structural elements of the inputs: parse

- **Question (CFQ input)**

```
Did M0 ' s writer , editor , director , and cinematographer found M1 and found M2
```

- **Positive query (CFQ output)**

```
SELECT count ( * ) WHERE {
 ?x0 film.cinematographer.film M0 .
 ?x0 film.director.film M0 .
 ?x0 film.editor.film M0 .
 ?x0 film.writer.film M0 .
 ?x0 organizations_founded~ M1 .
 ?x0 organizations_founded~ M2
}
→ Label: 1 (same)
```

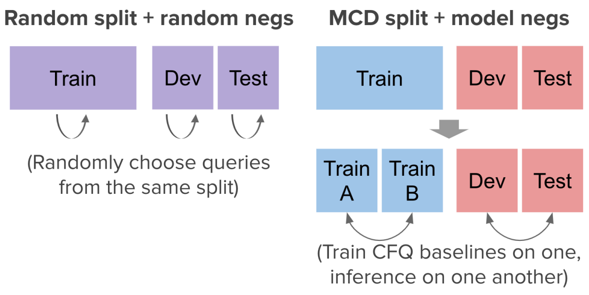
- **Model negative query**

```
SELECT count ( * ) WHERE {
 ?x0 film.cinematographer.film M0 .
 ?x0 film.director.film M0 .

 ?x0 film.writer.film M0 .
 ?x0 organizations_founded~ M1 .
 ?x0 organizations_founded~ M2
}
→ Label: 0 (different)
```

- **Random negative query**

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 a film.editor .
 ?x0 film.writer.film M1 .
 ?x0 film.writer.film M2 .
 ?x0 film.writer.film M3 .
 ?x0 ~.person.nationality m_0d060g
}

→ Label: 0 (different)
```

Figure 1: Examples of the CFQ classification dataset. Each query pairs with the question to form an instance. Note the model negative resembles the positive, while the random negative query differs considerably.



Random split + random negs
MCD split + model negs
Train
Dev
Test
(Randomly choose queries from the same split)
Train
Dev
Test
Train A
Train B
Dev
Test
(Train CFQ baselines on one, inference on one another)

Figure 2: Negative example strategies. Different colors indicate different compound distributions.

trees of both the natural language sentences and SPARQL queries, and *entity cross links* (linking entity mentions from the natural language side to the corresponding mentions in the SPARQL query).

The parse trees of the questions are already given in the original CFQ dataset as constituency-based parse trees. Since the trees include intermediate nodes indicating syntactic structures, we append tokens representing them at the end of each question. We created a simple parser to generate dependency-based parse trees for the SPARQL queries. We join the roots of the two trees to make a single global tree with the <CLS> token as the root.

We represent the structure of the inputs by masking attention ("hard mask") or with relative attention (Shaw et al., 2018) labels ("soft mask").

- Hard mask: We customize the binary attention mask of the original Transformer to only allow attention between tokens connected by the edges of the parse tree.

- Soft mask: For every pair of input tokens, we assign relative attention labels based on which of the following edge relationships applies: parent-to-child, child-to-parent, itself, from-or-to-root, or entity-cross-link.

Additionally, we allow attention pairs in the masks connecting the entities appearing both in the question and the queries. We call these links *entity cross links*, and they are found by simple string match (e.g. "M0"). Notice that while relative attention labels and the additional tokens to represent the constituency parse tree of the natural language add capabilities to the model, the "hard mask" structure annotations described above (which result in the larger performance gains) do not *add* any attention capabilities to the model. Instead, they simply remove non-structure attention edges. Figure 3b shows the parse trees, and Figure 3c and 3d show the masks for an example.

5 Results and Discussion

We used the ETC (Ainslie et al., 2020) Transformer model implementation as it allows us to provide the hard and soft masks described above in an easy way. In all experiments, we report AUC in the dev set as the evaluation metric (we did not evaluate on the test set). Please see Section A in the appendix for training details.

5.1 The CFQ Classification Dataset

We generate two classification datasets: "random split & random negatives" and "MCD split & model negatives", and evaluate LSTM and Transformer models. For both datasets, we evaluate AUC on the hold-out set (taken out of the training set as described above) to test i.i.d. generalization, and on the dev set to test compositional generalization.

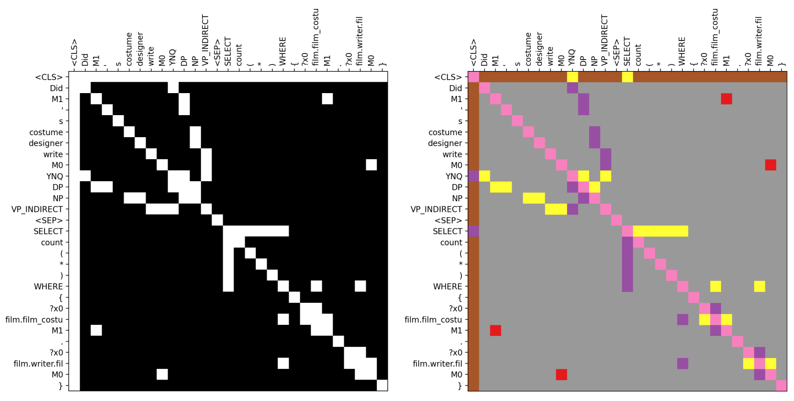
As shown in Table 1, models easily generalize on the hold-out set (AUC $\geq$ 0.99). All baseline models also achieve almost 1.0 AUC in the dev set of the "random split & random negatives". However, in the "MCD split & model negatives" models cannot generalize well on the dev set, showing compositional generalization is required. Note that random guessing achieves 0.5 AUC score.

```
Question: Did M1 ' s costume designer write M0
Query:
SELECT count ( * ) WHERE {
    ?x0 film.film_costumer_designer~ M1 .
    ?x0 film.writer.film M0
}
```

(a) A CFQ example

```
YNQ                SELECT
├Did               ├count
├DP                ├(
│├M1               ├*
│├'                ├)
│└NP               └WHERE
│ ├costume          ├film.film_costumer_designer~
│ └designer         │├?x0
└VP_INDIRECT        │└M1
 ├write             └film.writer.film
 └M0                 ├?x0
                     └M0
```

(b) Parse trees of the CFQ example



(c) Hard mask (d) Soft mask

Figure 3: Structure annotations for a CFQ example. We extract the hierarchical structure of the question and query of CFQ examples and use them to mask attention (hard mask) and/or provide relative attention labels (soft). Different colors indicate different relative attention labels.

Model	*Random Split & Random Neg*			*MCD Split & Model Neg*		
	Train	**Hold-out**	**Dev**	**Train**	**Hold-out**	**Dev**
LSTM	1.0000	0.9998	0.9998	1.0000	0.9972	0.8310
Transformer (2 layers)	0.9998	0.9997	0.9998	0.9988	0.9931	0.8789
Transformer (6 layers)	0.9999	1.0000	0.9999	0.9995	0.9931	0.8738

Table 1: AUC on the CFQ classification dataset generated with different methods

Model	**Mask Type**	**Cross link**	*MCD Split & Model Neg*		
			Train	**Hold-out**	**Dev**
LSTM	-		1.0000	0.9972	0.8310
Transformer	-		0.9995	0.9931	0.8738
Transformer w/ structure annotations (ETC)	No	-	0.9994	0.9934	0.8868
	Hard	N	0.9999	0.9978	0.9061
		Y	1.0000	0.9992	<u>0.9656</u>
	Soft		0.9995	0.9936	0.8819
	Both		1.0000	0.9991	**0.9721**

Table 2: AUC on the CFQ classification dataset (*MCD Split & Model Neg*) with various structure annotations

5.2 Structure Annotation

Table 2 compares different ablations of our structure annotation approach compared to the baseline models. The first (no masks and no cross links) just shows that adding tokens to the input to represent the constituency parsing and moving to ETC only provide small gains (from 0.8738 to 0.8868 AUC). Adding a hard mask already helps the model (0.9061 AUC), and adding cross links on top of that achieves very significant gains (0.9656 AUC). Finally, soft masks by themselves do not seem to help, but a combination of soft and hard masks achieves our best result of 0.9721 AUC.

The interesting result here is that adding the hard mask with entity cross links *only removes* potential attention pairs, so it does not increase model capacity in any way. In other words, the underlying transformer model is in principle able to generalize compositionally to some extent but seems to struggle in suppressing non-compositional attention.

6 Conclusions

The main contribution of this paper is to show that providing structure annotations in the form of attention masks significantly helps Transformer models generalize compositionally. This is interesting for two main reasons: first, it shows that neural network models do have the innate ability to generalize compositionally to some extent, but need some guidance to do so (e.g., by providing attention masks as in our work). This reinforces previous work showing that LSTMs also can, in principle, generalize compositionally, but they just do so with very low probability (Liška et al., 2018). The second reason is that structure annotations, which we provided manually, could be generated by another model in future work. We also presented a procedure for generating classification datasets that require some degree of compositional generalization starting from sequence generation datasets.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283.

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284.

Jacob Andreas. 2019. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2018. Universal transformers. *arXiv preprint arXiv:1807.03819*.

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*.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

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.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

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.

Adam Liška, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

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

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

A Full Experimental Results

In this section, we report the full results on the CFQ classification dataset and the structure annotation experiments. In all configurations, multiple evaluation metrics (accuracy, F1 score, and AUC) are computed by averaging the results of two randomly initialized experiments. We test each network using only the val set, not the test set, since the main purpose of the experiment is to compare the compositional generalization ability, not to select best hyper-parameter. Accuracy and F1 score are computed with the threshold 0.5 of the softmax output of label 1.

All the experiments of the CFQ classification datasets were run using the TensorFlow (Abadi et al., 2016) framework. As we explain in the Section 5, we use the ETC Transformer (Ainslie et al., 2020) code for relative position embeddings. For the Transformer implementation, we use the code provided in a Tensorflow tutorial. The training is run on the `n1-highmem-8` instance (52GB RAM, 8 virtual cpus) of Google Cloud Platform, extended with NVIDIA Tesla V100 GPUs.

Hyper-parameters used in the training of neural networks are listed in Table 3. One thing that we want to clarify is that training steps are required number of steps to converge and the training did not last longer than needed. Nevertheless, the experiments with structure annotations required more training steps than LSTM/Transformer, especially when the network is using hard mask. We conjecture that training with the hard mask of parse trees is slow since only a small part of the attention is not masked and hence propagating the gradient via supervision at the `<CLS>` position is slow.

A.1 The CFQ classification Dataset

Table 4 shows the classification results of various methods of generating classification datasets, including one additional configuration (*MCD Split & Random Negatives*). The dataset generated by this new configuration has the train and the dev/test set that have different compound distributions, because it is based on the MCD split. However, because of the method used in generating negative instances (random negatives), the binary classification of correspondence can be easily generalizable to the dev set.

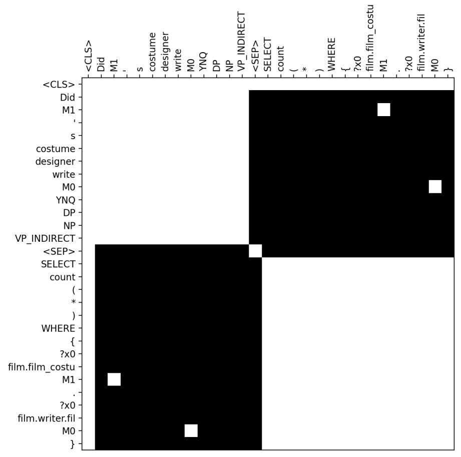


Figure 4: Block attention mask for the CFQ classification example of Figure 3. The dots at top-right and bottom-left are from entity cross links.

A.2 Structure Annotation

One possible annotation of the input structure is a mask to allow tokens of the question and SPARQL queries to only attend within their segment. We call this mask as *block attention* and test it as an alternative to the hierarchical attention structures (parse trees). This mask is denser than the attention mask from parse trees and sparser than "no mask". Figure 4 shows the block attention for the examples shown in the Figure 3.

Table 5 reports the full results of experiments on structure annotations. In all cases, entity cross links improve compositional generalization on the dev set, but provide a significant gain only when combined with the parse tree attention and the attention is guided by the "hard mask". As we can see in the "hard mask" experiments, block attention does not improve compositional generalization, which suggests a need for more detailed attention mask of input structure.

B Related Works on Compositional Generalization

In this section, we review prior works on improving compositional generalization in more detail.

Russin et al. (2019) proposed to split the attention mechanism into two separate parts, syntax and semantics. The semantic part encodes each token independent of the context (this is a pure embedding look-up table), and the syntactic part encodes each token by looking only at its context (without looking at the token itself). In this way, the syntactic part tries to capture the syntactic role a token

	LSTM	Transformer	ETC
Hidden layers	2	{2,6}	6
Last dense layers	2	1	1
Hidden Size	512	128	128
Filter size	-	2048	512
Number of heads	-	16	16
Dropout	0.4	0.1	0.1
Batch size	1024	512	112
Training steps			
Random & Random	20k	10k	-
MCD & Random	20k	10k	-
MCD & Model	30k	20k	200k
Optimizer	Adam (0.85, 0.997)	Adam (0.9, 0.997)	Adam (0.9, 0.997)
Learning rate schedule	Constant	Inverse sqrt	Inverse sqrt
Base learning rate	0.001	0.001	0.001
Warmup steps	-	1000	1000
Weight decay	0.0	0.0	0.0

Table 3: Hyper-parameters used in training deep neural networks on the CFQ classification datasets

might play in a sequence. They show improved compositional generalization on the SCAN dataset using LSTMs, with respect to using standard attention. Compared to Russin et al. (2019) that uses LSTMs for the syntactic part, we use Transformer architecture to handle the hierarchical structure of the input.

In their follow up work on the CFQ dataset, Furrer et al. (2020) showed that an increased amount of pre-training helped Transformer models better generalize compositionally.

Another idea that has been proposed is to augment the training data, adding synthetic training examples to give the model a compositional learning bias (Andreas, 2019) .

Finally, work also exists on using general-purpose models like *Neural Turing Machines* or *Differential Neural Computers* (Graves et al., 2016) that are often trained via reinforcement learning to solve compositional generalization tasks. These models learn an “algorithm” that can solve the task at hand, rather than trying to learn a direct input/output mapping as the Transformer models used in most other works do.

C Examples of the CFQ classification dataset

In Figure 5, we present more examples of the CFQ classification datasets. In all cases, the random negative queries substantially differ from the positive queries, implying that a learner can easily perform the task. On the other hand, the model negative queries only differ by a token or a phrase, which demands a learner’s higher ability.

Dataset 1: *Random Split & Random Negatives*

Model	**Train**			**Train (hold-out)**			**Dev**		
	Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC
LSTM	0.9999	0.9998	1.0000	0.9984	0.9967	0.9998	0.9982	0.9964	0.9998
Transformer (2 layers)	0.9988	0.9976	0.9998	0.9982	0.9964	0.9997	0.9988	0.9975	0.9998
Transformer (6 layers)	0.9992	0.9988	0.9999	0.9989	0.9978	0.9999	0.9990	0.9979	0.9999

Dataset 2: *MCD Split & Random Negatives*

Model	**Train**			**Train (hold-out)**			**Dev**		
	Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC
LSTM	0.9999	0.9998	1.0000	0.9982	0.9965	0.9999	0.9546	0.9025	0.9923
Transformer (2 layers)	0.9982	0.9965	1.0000	0.9974	0.9948	0.9999	0.9942	0.9883	0.9996
Transformer (6 layers)	0.9986	0.9972	0.9999	0.9979	0.9958	0.9997	0.9889	0.9775	0.9991

Dataset 3: *MCD Split & Model Negatives*

Model	**Train**			**Train (hold-out)**			**Dev**		
	Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC
LSTM	0.9990	0.9979	1.0000	0.9796	0.9604	0.9972	0.8226	0.5199	0.8310
Transformer (2 layers)	0.9817	0.9639	0.9988	0.9592	0.9202	0.9931	0.8359	0.5835	0.8789
Transformer (6 layers)	0.9886	0.9776	0.9995	0.9582	0.9189	0.9931	0.8414	0.6191	0.8738

Table 4: Results of the CFQ classification dataset generated with different CFQ splits and negative example strategies

Model	**Mask Type**	**Parse Tree**	**Block Attn**	**Cross link**	**Train**			**Train (hold-out)**			**Dev**		
					Acc	F1	AUC	Acc	F1	AUC	Acc	F1	AUC
LSTM	-				0.9990	0.9979	1.0000	0.9796	0.9604	0.9972	0.8226	0.5199	0.8310
Transformer	-				0.9886	0.9776	0.9995	0.9582	0.9189	0.9931	0.8414	0.6191	0.8738
Transformer w/ structure annotations (ETC)	No	-			0.9874	0.9751	0.9994	0.9591	0.9199	0.9934	0.8434	0.6202	0.8868
	Hard	Y	N	N	0.9955	0.9911	0.9999	0.9766	0.9543	0.9978	0.8628	0.6744	0.9061
		Y	N	Y	0.9978	0.9956	1.0000	0.9866	0.9738	0.9992	0.9170	0.8269	0.9656
		N	Y	N	0.9828	0.9659	0.9989	0.9567	0.9152	0.9928	0.8324	0.5874	0.8771
		N	Y	Y	0.9871	0.9746	0.9993	0.9573	0.9171	0.9930	0.8386	0.6048	0.8881
	Soft	Y	N	N	0.9863	0.9728	0.9993	0.9588	0.9197	0.9933	0.8426	0.6017	0.8729
		Y	N	Y	0.9891	0.9784	0.9995	0.9603	0.9226	0.9936	0.8482	0.6385	0.8819
	Hard +Soft	Y	N	N	0.9940	0.9882	0.9999	0.9743	0.9500	0.9973	0.8615	0.6697	0.9056
		Y	N	Y	0.9975	0.9949	1.0000	0.9867	0.9739	0.9991	0.9249	0.8473	**0.9721**

Table 5: Results of the CFQ classification dataset (MCD split & model negatives) with different types of structure annotations

• Question (CFQ input)

```
Did M0 ' s founder produce M1
```

• Positive query (CFQ output)

```
SELECT count ( * ) WHERE {
 ?x0 film.producer.film~ M1 .
 ?x0 ~organizations_founded M0
}

→ Label: 1 (same)
```

• Model negative query

```
SELECT count ( * ) WHERE {
 ?x0 film.producer.film~ M0 .
 ?x0 ~organizations_founded M1
}

→ Label: 0 (different)
```

• Random negative query

```
SELECT count ( * ) WHERE {
 ?x0 a film.film .
 ?x0 film.film.edited_by ?x1 .
 ?x0 film.film.edited_by M3 .
 ?x0 film.film.edited_by M4 .
 ?x0 film.film.written_by M1 .
 ?x0 film.film.written_by M2 .
 ?x1 a film.actor
}
→ Label: 0 (different)
```

(a)

• Question (CFQ input)

```
Did a British sibling of M0 direct M2
```

• Positive query (CFQ output)

```
SELECT count ( * ) WHERE {
 ?x0 film.director.film M2 .
 ?x0 ~.person.nationality m_07ssc .
 ?x0 people.person.sibling_s~ M0 .
 FILTER ( ?x0 != M0 )
}
→ Label: 1 (same)
```

• Model negative query

```
SELECT count ( * ) WHERE {
 ?x0 film.director.film M2 .
 ?x0 ~person.nationality m_07ssc .
 ?x0 people.person.sibling_s~ M2 .
 FILTER ( ?x0 != M0 )
}
→ Label: 0 (different)
```

• Random negative query

```
SELECT count ( * ) WHERE {
 ?x0 a film.editor .
 ?x0 film.cinematographer.film M3 .
 ?x0 ~.person.gender m_05zppz .
 ?x0 ~.person.nationality m_03_3d
}
→ Label: 0 (different)
```

(b)

• Question (CFQ input)

```
Which male person directed M2 , M3 , and M4
```

• Positive query (CFQ output)

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 a people.person .
 ?x0 film.director.film M2 .
 ?x0 film.director.film M3 .
 ?x0 film.director.film M4 .
 ?x0 people.person.gender m_05zppz
}

→ Label: 1 (same)
```

• Model negative query

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 a people.person .
 ?x0 film.director.film M2 .
 ?x0 film.director.film M3 .
 ?x0 film.director.film M4 .
 ?x0 people.person.gender m_02zsn
}

→ Label: 0 (different)
```

• Random negative query

```
SELECT count ( * ) WHERE {
 ?x0 a film.actor .
 ?x0 film.editor.film M1 .
 ?x0 film.editor.film M2 .
 ?x0 ~films_executive_produced M3 .
 ?x0 film.writer.film ?x1 .
 ?x1 a film.film
}
→ Label: 0 (different)
```

(c)

• Question (CFQ input)

```
Who directed a film , executive produced M1 and M2 , and executive produced M3 and M4
```

• Positive query (CFQ output)

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 a people.person .
 ?x0 film.director.film ?x1 .
 ?x0 ~films_executive_produced M1 .
 ?x0 ~films_executive_produced M2 .
 ?x0 ~films_executive_produced M3 .
 ?x0 ~films_executive_produced M4 .
 ?x1 a film.film
}

→ Label: 1 (same)
```

• Model negative query

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 a people.person .
 ?x0 ~films_executive_produced ?x1 .
 ?x0 ~films_executive_produced M1 .
 ?x0 ~films_executive_produced M2 .
 ?x0 ~films_executive_produced M3 .
 ?x0 ~films_executive_produced M4 .
 ?x1 a film.film
}

→ Label: 0 (different)
```

• Random negative query

```
SELECT count ( * ) WHERE {
 ?x0 a film.cinematographer .
 ?x0 film.editor.film M1 .
 ?x0 film.editor.film M2 .
 ?x0 film.editor.film M3 .
 ?x0 ~films_executive_produced M1 .
 ?x0 ~films_executive_produced M2 .
 ?x0 ~films_executive_produced M3 .
 ?x0 film.writer.film M1 .
 ?x0 film.writer.film M2 .
 ?x0 film.writer.film M3
}
→ Label: 0 (different)
```

(d)

Figure 5: Examples of the CFQ classification dataset. Each query pairs with the question to form an instance. Note the model negative resembles the positive, while the random negative query differs considerably. In the model negative queries, the differences from the positive query are marked in bold.