

Revisiting the Effects of Leakage on Dependency Parsing

Nathaniel Krasner^{*,1}, Miriam Wanner^{*,2}, Antonios Anastasopoulos¹

¹Department of Computer Science, George Mason University

²Department of Computer Science, University of Virginia

natekrasner@gmail.com, msw2vg@virginia.edu, antonis@gmu.edu

Abstract

Recent work by Søgaaard (2020) showed that, treebank size aside, overlap between training and test graphs (termed *leakage*) explains more of the observed variation in dependency parsing performance than other explanations. In this work we revisit this claim, testing it on more models and languages. We find that it only holds for zero-shot cross-lingual settings. We then propose a more fine-grained measure of such leakage which, unlike the original measure, not only explains but also correlates with observed performance variation.¹

1 Introduction

Syntactic parsing has long been one of the core natural language processing (NLP) tasks, and the proliferation of the Universal Dependencies project (UD; de Marneffe et al., 2021; Nivre et al., 2017) has allowed the development and comparison of monolingual and multilingual models under the same syntactic framework.

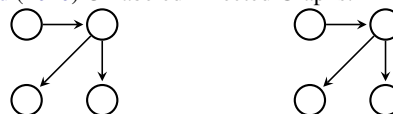
The performance of the dependency parsers, however, varies wildly across languages, with state-of-the-art performance ranging from labeled attachment scores below 20 (e.g. for Amharic, Erzya, Komi, or Yoruba) to more than 90 (e.g. for Spanish, Polish, Russian, or Greek). As the UD treebanks follow mostly similar annotation guidelines, comparisons of the parsing performance across languages are now possible, to an extent.²

In an effort to explain these cross-lingual performance differences, researchers have proposed treebank size (Vania et al., 2019), linguistic variation (Nivre et al., 2007), test data sentence length

Dependency Trees:



Søgaaard (2020) Unlabeled Directed Graphs:



Node- and Edge-Labeled Directed Graphs:

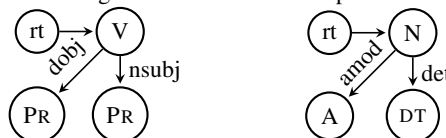


Figure 1: Only labeled reductions produce different graphs for these fundamentally different sentences. Under unlabeled leakage, the two trees do leak. When taking labels into account the two trees belong to different isomorphisms and are not considered “leaky”.

or average gold dependency length (McDonald and Nivre, 2011), and domain differences between training and test data (Foster et al., 2011), as potential predictors. Recently, Søgaaard (2020) proposed that the proportion of isomorphic graph structures between the training and testing data (*leakage*) is a stronger predictor of the parsers’ performance than any of the previously listed attributes other than training treebank size.

Søgaaard (2020) concludes that “some languages seem easier to parse because their treebanks leak.” This finding is potentially crucial for current parser evaluation on the existing treebanks, as well as for future treebank construction. It implies, for instance, that parsers are perhaps not as good as they seem, because they are tested on “leaky” test data. Perhaps one should also consider designing treebanks that do not leak between train and test, as such a test set would not have a bias toward more common phenomena.

In this work, we examine this finding more closely. We extend Søgaaard’s definition to include

* Equal contribution. Work performed at GMU.

¹Code and data are available here: <https://github.com/miriamwanner/reu-nlp-project>

²Different treebank creation protocols followed across languages (whose effects are hard to isolate or measure) can be a significant source of variation. Nevertheless, *some* of the observed variation can be possibly explained by other factors. We direct the reader to footnote 2 of (Søgaaard, 2020).

labeled leakage, and study it over multiple parsers in both monolingual and cross-lingual settings. We show that the finding does not hold up when tested against more modern parsers and more languages. We do identify, though, that leakage indeed predicts parser performance in zero-shot cross-lingual settings, and we dive deeper in this phenomenon with an extensive study focusing on Faroese and other Germanic languages. Last, we propose a modification of the leakage measure that both predicts *and correlates with* parser performance in such settings.

2 Leakage and How to Measure it

In this section we first define leakage based on graph isomorphisms and reproduce Søgaaard’s experiments. We then show that parsers make *local* decisions that allow them to generalize to unseen graphs, and explore additional measures of leakage, studying whether they help explain parser performance. Last, we argue that sub-trees are more meaningful units than label-free, tree-level representations.

Leakage Definition Leakage can be broadly defined as the portion of test trees that have isomorphic counterparts in the train set. While dependency trees are labeled, directed graphs with labels both on the nodes and on the edges, Søgaaard (2020) performed a reduction by removing labels from both nodes and edges.

Given these reduced graphs, Søgaaard (2020) finds the different isomorphisms that are present in the training and the test set, using the VF2 algorithm (Cordella et al., 2001). We note that the isomorphism may or may not rely on node or edge labels. In the experiments below, we perform an ablation between using completely unlabeled directed graphs, node-labeled (but not edge-labeled) directed graphs, and between using the full information of the graphs to compute isomorphisms, namely both node and edge labels.

Reproducing (Søgaaard, 2020) Examples of the reductions needed for computing leakage for two sentences are shown in Figure 1. Now, assume that the first sentence is in the training set and the second is part of the test set. Measuring leakage without labels implies that the first dependency tree is somehow informative for producing the tree for the second sentence, which we believe is counter-intuitive. Hence, our first hypothesis is that a more informed leakage calculation is going to explain

more of the performance variance.

We reproduce the experiments of Søgaaard (2020) comparing the three different reductions (denoted as "none" for unlabeled graphs, "edges" and "nodes+edges" for respectively labeled graphs). The experiment consists of correlating the factors ϕ assumed to influence syntactic dependency parser performance with the performance of the parser under study. We train a simple linear regression model³ with treebank size and ϕ as input and parser performance as output. ϕ will correspond to our measure of treebank leakage. Mathematically, we have $\alpha t_s + \beta \phi + \gamma$ with t_s treebank size and α, β, γ learned parameters. Following Søgaaard, we will focus on explained variance and mean absolute error (MAE) from five-fold cross-validation to avoid overfitting. Unlike Søgaaard, we will additionally report Spearman’s ρ correlation coefficients⁴ between factor and performance, which will reveal whether indeed leakage leads to better parser performance.⁵

The results on the same data as Søgaaard (2020) (using the best reported parser performance from the CoNLL 2018 shared task) are presented in the top three rows of Table 1. We find that unlabeled graph leakage produces positive explained variance, in line with previous work. However, we have to reject our hypothesis, as a more informed leakage measurement fails to meaningfully explain the output variance, producing negative scores. In fact, the more information we use when computing the graph isomorphisms, the less the model can explain output variance!⁶

To further solidify this finding, we repeat the above experiment, this time using UDify, the state-of-the-art multilingual parser of Kondratyuk and Straka (2019).⁷ The result is shown in the bottom

³Exactly as Søgaaard (2020) does, just on different data/settings.

⁴We do not expect the correlation, if any, to be linear, hence we prefer Spearman’s measure to Pearson’s.

⁵Note that the explained variance is basically the correlation squared. As such, it cannot reveal whether the correlation is positive or negative. Negative explained variance means that the model is a poor fit for the data (worse than just predicting the average).

⁶Søgaaard (2020) gives this possible explanation: “The result is perhaps not too surprising, since graph isomorphisms correlate with syntactic constructions, which in turn correlate with the occurrence of linguistic markers and tail linguistic phenomena.”

⁷The model is trained jointly on all UD treebanks (that have a training set), and hence in this experiment we compute leakage multilingually (i.e. we compute leakage between the complete training set and the test set of each treebank).

Leakage Attributes	Tree-level leakage			
	Regression Score	Expl. Variance	MAE	Spearman's ρ
System: CoNLL'18 (Søgaard)				
None	0.162	0.143	7.257	-0.194
Edges	0.091	-0.121	8.592	-0.181
Nodes+Edges	0.085	-0.179	8.830	-0.161
System: UDify				
None	0.250	0.047	13.07	-0.360
Edges	0.146	-0.083	14.012	-0.080
Nodes+Edges	0.134	-0.108	14.156	-0.026

Leakage Attributes	Sub-tree-level leakage			
	Regression Score	Expl. Variance	MAE	Spearman's ρ
System: CoNLL'18 (Søgaard)				
None	0.054	-0.137	8.248	-0.238
Edges	0.083	-1.202	9.444	0.390
Nodes+Edges	0.089	-0.210	8.197	0.538
System: UDify				
None	0.123	-0.171	14.632	-0.082
Edges	0.174	-0.333	14.182	0.579
Nodes+Edges	0.217	-0.149	13.715	0.654

Table 1: Tree-level leakage (left) does not correlate with and does not always explain parser performance. Labeled sub-tree level leakage (right) however is positively correlated with parser performance.

Construction	Actual	Produced by model trained without	
		nsubj mods	obj mods
nsubj mods	3166	1698	3167
obj mods	3910	4505	1940

Table 2: Number of adjectival modifiers produced by counterfactual models. The parsers can produce constructions not seen during training.

three rows of Table 1 (left Table), and they present more negative evidence for our hypothesis: there is minimal explained variance in the unlabeled version, and still negative explained variance in the labeled leakage versions.

Hence, we have to –for now– reject our hypothesis: using labeled graph isomorphisms to compute leakage does not explain more downstream parser performance variations, at least when using tree-level leakage measures; we revisit this hypothesis for sub-tree leakage below. Concurrently, we need to highlight the fact that for *all* cases we focused on this experiment, there was a negative (inverse) correlation between leakage and parser performance.

While Søgaard (2020) was correct (for the languages/parsers they studied) to state that there is a correlation between leakage and parser performance, we believe they reached an incorrect conclusion. The metric they used (explained variance) does not reveal the direction of the correlation, just that there is a correlation. Because of this they came to the wrong conclusion that there was a positive correlation between leakage and parser performance. Our results (Table 1, left table) instead imply that as leakage increases, parser performance worsens! Clearly, something is wrong and we need to re-examine Søgaard’s reasoning.

Sub-Trees are More Meaningful Units We turn our attention to the parsers whose performance we are trying to explain.

The three parsers that Søgaard uses and UDify are graph-based ones. This means that they do not necessarily score or produce whole trees. Graph-based parsers score pairs of words, and from these scores a minimum spanning tree is selected to produce the final dependency parse. As such, we argue that whole trees as a measure of leakage are not appropriate for graph-based parsers.

To drive this point forward, we perform synthetic experiments removing adjectival modifiers from nominal subject or object. In particular, we created training data in which the data either did not contain adjectival modifiers on subject/object nouns. We then tested the models on gold unmodified test data which contained such modifiers. By removing adjectival modifiers only from the subjects (similarly for objects) in the training data, we ensure two things: that test instances with adjectival modifiers on subjects are not leaky; as such, if whole-tree leakage is a proper indication for parser performance, then the parser should perform poorly in producing such constructions.

Table 2 shows the results of our experiment over the German HDT treebank.⁸ We found that the parsers trained on our counterfactual data, which have zero leakage for these test instances, still produce the local constructions that they have never observed during training. The parsers trained without training subject modifiers produced about half of the expected subject modifiers (similarly for object modifier experiments). Nevertheless, they were still able to generalize based on other similar constructions seen in training, correctly parsing a non-zero amount of unseen-in-training constructions.

This observation is not unexpected. In the experiment above, even removing all adjectival modifiers from nouns that are subjects (hence a subtree –and

⁸Also see results on English in Appendix B.

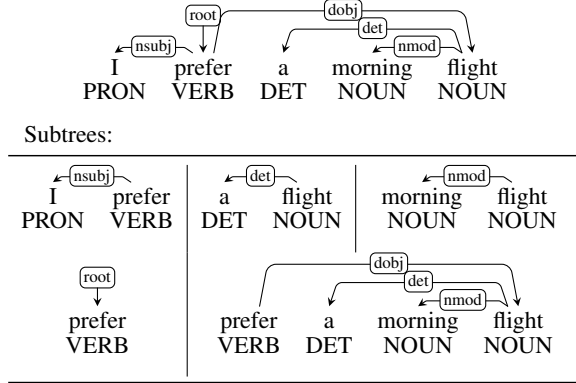


Figure 2: Decomposition of a tree (top) into a set of sub-trees (bottom).

consequently a whole tree containing it— has never been observed), the parser still observes adjective-noun modifying pairs elsewhere in the sentences and is able to generalize, producing a tree that has never been observed at training.

The fact that the parsers make *local* output decisions, along with the proven corollary that they can easily produce unseen trees, guides us to search for a leakage measure focusing on sub-trees.

Sub-Tree Based Leakage We define a leakage measure where a dependency tree is first decomposed into a set of subtrees, and then each subtree reduces into the graphs defined above to compute isomorphisms. These subtrees are created for each node (word), connecting it to its parent and to all its children. See example in Figure 2.

We repeat our experiments, this time using our proposed leakage measure, and present the results in the right-hand side of Table 1. As before, for Søggaard’s and for the UDify combinations of models/languages the explained variance is negative. However, now more information (edge/node labels) leads to higher Spearman’s ρ coefficients, implying that indeed the more test subtrees we have observed in training, the better the parser performance.

In the unlabeled setting, every sub-tree created by the parser was found in the training data, which was true of most gold files as well. We interpret this observation to mean that *unlabeled* sub-trees are not meaningful units, a point further reinforced by the negative explained variance and correlations.

At the same time, our measure still fails to explain any of the observed performance variance. Thus, we have to reach a conclusion opposite of Søggaard (2020), that **in a monolingual setting** the performance of modern graph-based parsers is not particularly explained by train-test leakage, however we compute that leakage.

System: UDify (Zero-Shot)	Regression Score	Expl. Variance	MAE	Spearman’s ρ
Whole-Tree Leakage				
None	0.385	0.263	17.539	-0.581
Edges	0.221	0.124	21.460	-0.493
Nodes+Edges	0.221	0.106	21.506	-0.546
Sub-Tree Leakage				
Edges	0.271	0.210	18.245	0.609
Nodes+Edges	0.246	0.215	18.038	0.578

Table 3: Leakage explains zero-shot parser performance. Sub-tree leakage also correlates with it.

3 Leakage Explains 0-Shot Performance

Modern dependency parsing models trained on many languages perform well on languages unseen (zero-shot setting) during training (Muller et al., 2021; Glavaš and Vulić, 2021, *inter alia*).

We focus again on UDify, since it performs well in zero-shot settings. This is generally attributed to two factors: the presence of related languages in the training set, and the multilingual capabilities of the underlying representation model (here, a multilingual BERT (Devlin et al., 2019) model).

Table 3 reports results with Søggaard’s (whole-tree) and our sub-tree leakage measures under all three settings, focusing only on 35 zero-shot test languages.⁹ Now leakage¹⁰ can indeed explain downstream parser performance. Our proposed measure explains as much variance as the original whole-tree measure *and also* correlates with performance.

Analysis on Faroese Looking deeper into the zero-shot setting, we perform an experiment on a simplified bilingual zero-shot setting. We train parsers in five Germanic languages (German, Swedish, Danish, Norwegian, Icelandic) and test on Faroese in a zero-shot fashion. For each language, we train a model on:

- a ‘leaky’ sample of the portion of training treebank, so that all training data overlapped with (some) test data,
- a ‘non-leaky’ sample of trees such that there was no train-test overlap,
- a control random sample from the training treebank, and
- a ‘diverse’ training sample including a single tree from each isomorphism equivalence class.

⁹These 35 languages are treebanks with size 0 in Konratyuk and Straka (2019).

¹⁰In this multilingual setting, leakage is computed against a training set comprised of 75 concatenated treebanks.

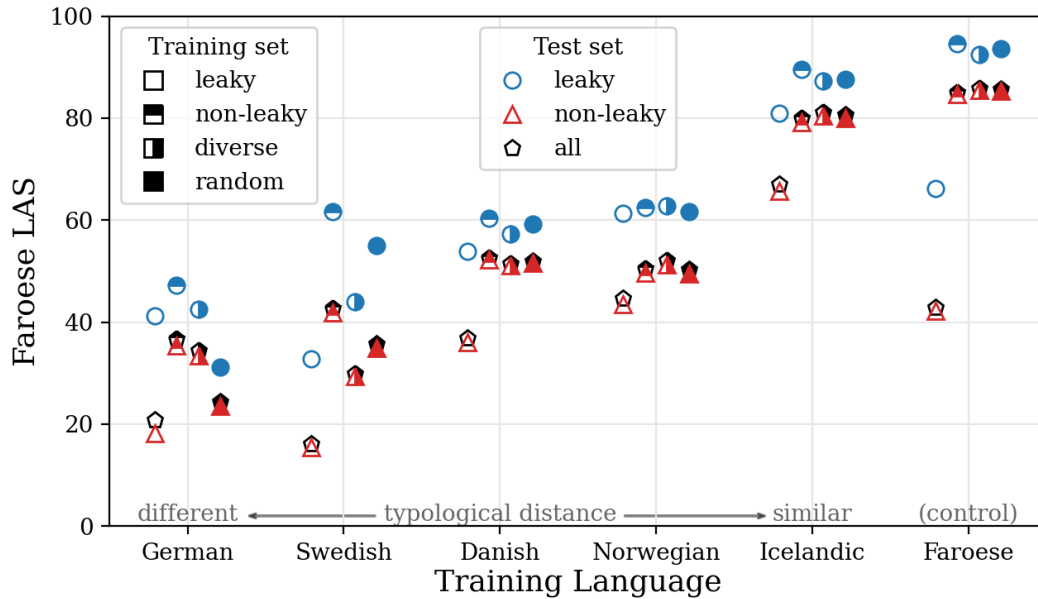


Figure 3: Zero-shot results on Faroese. Training on non-leaky and diverse data is best. The leaky portion of the test set is far easier than the rest.

All of the above models are size-controlled for each language, so that the training data sizes are exactly the same. Leakage here is measured with unlabeled full-tree leakage, for simplicity. We similarly split the test set, for each language, into ‘leaky’ and ‘non-leaky’ subsets (also reporting numbers for the whole test set).

For example, take German training and Faroese test sets. First all German instances leaking into Faroese are added to the “German leaky” train set and the corresponding leaked Faroese sentences are put into the “Faroese-leaky” test set. Then the remaining sentences from the German training set are added to the “German nonleaky” train set and the remaining sentences from the Faroese test set are the “Faroese nonleaky” test set. Last, we take a random sample of same size across all settings, so that training data size is not a confounding factor for our analysis.

See Figure 3 and extensive results in Appendix C. For all languages, models trained on leaky data perform worse than models trained on the same amount of non-leaky or random data. For most transfer languages, in fact, training solely on non-leaky data performs better than training on other subsets! In addition, the leaky part of the testing data is clearly *easier* to parse in general, while the non-leaky part is more challenging.

The models trained on perfectly ‘diverse’ treebanks generally perform just as good as those trained on non-leaky or randomly sampled data

and often better on the non-leaky test set, which means they generalize better. This indicates a way to reach better cross-lingual performance without the need for large training data, as long as the training set is diverse enough.

The large performance difference between models trained on leaky and non-leaky trees reveals that something is different about the parts of the treebanks that leak. We measured the diversity of the leaky, non-leaky, and randomly selected trees, defined as the number of unique trees divided by the total number of trees. We found that leaky treebanks were far less diverse and therefore contain fewer unique structures than non-leaky or randomly sampled counterparts. Across all treebanks, leaky instances are also generally shorter (8.4 vs 21.6 avg length), shallower (2.2 vs 4.8 average tree depth), and with shorter avg dependency length (2 vs 3.3).

We argue that the reasoning should be reverse: short “easy” examples are more likely to leak; it is not leakage that makes them easy!

4 Conclusion

We re-evaluate claims that training-test leakage can explain parser performance, define a subtree-based leakage measure that better explains performance, and show that this claim only holds for zero-shot transfer settings.

Acknowledgements

The authors express their gratitude to Djamé Seddah for very helpful initial discussions, and to the reviewers for their constructive feedback. The first two authors were supported in part by the NSF IIS-1757064 Undergraduate Research in Educational Data Mining grant. Antonios Anastasopoulos is also generously supported by NSF grants BCS-2109578 and IIS-2125466.

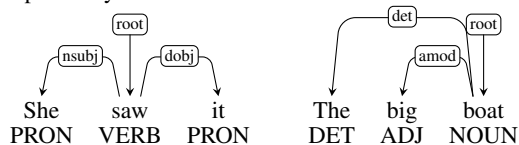
References

- Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2001. An improved algorithm for matching large graphs.
- Marie-Catherine de Marneffe, Christopher D Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal dependencies. *Computational linguistics*, 47(2):255–308.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.
- Jennifer Foster, Özlem Çetinoğlu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. 2011. [From news to comment: Resources and benchmarks for parsing the language of web 2.0](#). In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 893–901, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.
- Goran Glavaš and Ivan Vulić. 2021. [Climbing the tower of treebanks: Improving low-resource dependency parsing via hierarchical source selection](#). In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4878–4888, Online. Association for Computational Linguistics.
- Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795.
- Ryan McDonald and Joakim Nivre. 2011. [Analyzing and integrating dependency parsers](#). *Computational Linguistics*, 37(1):197–230.
- Benjamin Muller, Antonios Anastasopoulos, Benoît Sagot, and Djamé Seddah. 2021. [When being unseen from mBERT is just the beginning: Handling new languages with multilingual language models](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 448–462, Online. Association for Computational Linguistics.
- Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, et al. 2017. Universal dependencies 2.1.
- Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. [The CoNLL 2007 shared task on dependency parsing](#). In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics.
- Anders Søgaard. 2020. [Some languages seem easier to parse because their treebanks leak](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2765–2770, Online. Association for Computational Linguistics.
- Clara Vania, Yova Kementchedjhiava, Anders Søgaard, and Adam Lopez. 2019. [A systematic comparison of methods for low-resource dependency parsing on genuinely low-resource languages](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1105–1116, Hong Kong, China. Association for Computational Linguistics.

A Graph Reduction Examples

Shown in Table 4.

Dependency Trees:



Søgaard (2020) Unlabeled Directed Graphs:



Node-Labeled Directed Graphs:



Node- and Edge-Labeled Directed Graphs:

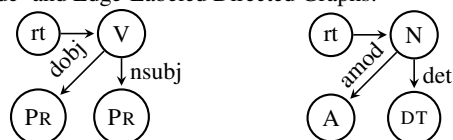


Figure 4: Only labeled reductions produce different graphs for these fundamentally different sentences.

B English Counterfactual Experiments

Shown in Table 4.

Construction	Actual	Produced by model trained without	
		nsubj mods	obj mods
German HDT treebank:			
nsubj mods	3166	1698	3167
obj mods	3910	4505	1940
English EWT treebank:			
nsubj mods	132	122	130
obj mods	254	258	237

Table 4: Number of adjectival modifiers produced by counterfactual models compared to the actual number in the gold file. The parsers can produce constructions not seen during training.

C Complete Faroese Results

Shown in Tables 5.

Train	Test	UAS	LAS	CLAS	MLAS	BLEX
Faroese-leaky	Faroese-leaky	78.48	66.24	55.6	51.26	55.6
	Faroese-nonleaky	53.22	42.19	28.39	22.6	28.39
	Faroese-all	53.91	42.85	29.19	23.43	29.19
Faroese-nonleaky	Faroese-leaky	97.89	94.51	90.65	90.65	90.65
	Faroese-nonleaky	88.15	84.75	78.66	75.98	78.66
	Faroese-all	88.42	85.02	79.01	76.41	79.01
Faroese-all	Faroese-leaky	97.47	93.67	90.32	88.17	90.32
	Faroese-nonleaky	89	85.36	79.02	76.56	79.02
	Faroese-all	89.23	85.59	79.35	76.9	79.35
Faroese-diverse	Faroese-leaky	96.62	92.41	89.61	85.3	89.61
	Faroese-nonleaky	89.27	85.56	79.66	77.2	79.66
	Faroese-all	89.47	85.75	79.95	77.43	79.95
German-leaky	Faroese-leaky	57.49	41.23	43	34.89	43
	Faroese-nonleaky	32.53	18.24	16.24	12.57	16.24
	Faroese-all	35.25	20.74	19.27	15.1	19.27
German-nonleaky	Faroese-leaky	64.08	47.18	51.43	40.86	51.43
	Faroese-nonleaky	52.07	35.43	35.14	27.09	35.14
	Faroese-all	53.38	36.71	36.98	28.65	36.98
German-all	Faroese-leaky	57.39	31.24	36.84	25.82	36.84
	Faroese-nonleaky	45.81	23.59	25.03	19.81	25.03
	Faroese-all	47.07	24.42	26.36	20.49	26.36
German-diverse	Faroese-leaky	58.02	42.51	48.94	29.73	48.94
	Faroese-nonleaky	48.95	33.4	34.26	23.6	34.26
	Faroese-all	49.94	34.39	35.9	24.29	35.9
Afrikaans-leaky	Faroese-leaky	46.51	20.47	24.91	11.72	24.91
	Faroese-nonleaky	27.19	7.49	7.04	3.88	7.04
	Faroese-all	27.67	7.81	7.54	4.1	7.54
Afrikaans-nonleaky	Faroese-leaky	47.44	16.28	7.43	5.57	7.43
	Faroese-nonleaky	39.78	15.99	6.96	5.47	6.96
	Faroese-all	39.97	16	6.97	5.47	6.97
Afrikaans-all	Faroese-leaky	45.58	17.21	22.8	10.42	22.8
	Faroese-nonleaky	40.51	16.34	10.03	5.82	10.03
	Faroese-all	40.64	16.36	10.37	5.94	10.37
Afrikaans-diverse	Faroese-leaky	38.6	16.74	13.04	9.94	13.04
	Faroese-nonleaky	36.18	16.34	8.13	6.48	8.13
	Faroese-all	36.24	16.35	8.26	6.57	8.26

Train	Test	UAS	LAS	CLAS	MLAS	BLEX
Danish-leaky	Faroese-leaky	71.33	53.85	55.73	47.83	55.73
	Faroese-nonleaky	49.86	35.97	34.74	27.31	34.74
	Faroese-all	50.93	36.86	35.86	28.41	35.86
Danish-nonleaky	Faroese-leaky	73.43	60.37	65.05	57.37	65.05
	Faroese-nonleaky	64.1	52.26	47.85	42.12	47.85
	Faroese-all	64.57	52.66	48.77	42.94	48.77
Danish-all	Faroese-leaky	69.46	59.21	64.08	54.69	64.08
	Faroese-nonleaky	62.52	51.63	48.34	42.12	48.34
	Faroese-all	62.86	52	49.18	42.79	49.18
Danish-diverse	Faroese-leaky	68.53	57.34	62.75	52.23	62.75
	Faroese-nonleaky	62.92	51.13	48.3	41.82	48.3
	Faroese-all	63.2	51.43	49.08	42.38	49.08
Icelandic-leaky	Faroese-leaky	88.07	80.97	74.1	69.54	74.1
	Faroese-nonleaky	72.77	65.78	53.78	50.26	53.78
	Faroese-all	74.02	67.02	55.54	51.93	55.54
Icelandic-nonleaky	Faroese-leaky	93.32	89.49	84.75	83.55	84.75
	Faroese-nonleaky	84.18	79.21	70.54	67.5	70.54
	Faroese-all	84.93	80.04	71.77	68.9	71.77
Icelandic-all	Faroese-leaky	91.05	87.64	82.93	80.53	82.93
	Faroese-nonleaky	84.27	79.97	71.36	68.49	71.36
	Faroese-all	84.82	80.6	72.37	69.54	72.37
Icelandic-diverse	Faroese-leaky	91.76	87.36	82.07	80.39	82.07
	Faroese-nonleaky	84.89	80.49	72.7	69.88	72.7
	Faroese-all	85.45	81.05	73.51	70.79	73.51
Norwegian-leaky	Faroese-leaky	72.46	61.41	65.53	61.8	65.53
	Faroese-nonleaky	52.78	43.48	36.09	32.2	36.09
	Faroese-all	54.04	44.62	38.09	34.21	38.09
Norwegian-nonleaky	Faroese-leaky	73.01	62.5	67.39	62.42	67.39
	Faroese-nonleaky	59.37	49.59	47	43.19	47
	Faroese-all	60.24	50.42	48.4	44.51	48.4
Norwegian-all	Faroese-leaky	73.01	61.59	65.54	59.69	65.54
	Faroese-nonleaky	61.12	49.49	45.64	42.46	45.64
	Faroese-all	61.88	50.27	47.01	43.64	47.01
Norwegian-diverse	Faroese-leaky	70.83	62.86	67.08	62.11	67.08
	Faroese-nonleaky	61.12	51.3	48.71	45.04	48.71
	Faroese-all	61.74	52.04	49.96	46.21	49.96

Train	Test	UAS	LAS	CLAS	MLAS	BLEX
Swedish-leaky	Faroese-leaky	60.58	32.75	38.81	27.29	38.81
	Faroese-nonleaky	42.92	15.44	17.44	14.72	17.44
	Faroese-all	43.63	16.13	18.33	15.24	18.33
Swedish-nonleaky	Faroese-leaky	75.07	61.74	64.22	58.33	64.22
	Faroese-nonleaky	57.22	41.87	38.77	33.37	38.77
	Faroese-all	57.94	42.67	39.83	34.41	39.83
Swedish-all	Faroese-leaky	68.99	55.07	57.35	49.76	57.35
	Faroese-nonleaky	49.95	34.85	31.8	26.79	31.8
	Faroese-all	50.71	35.65	32.86	27.74	32.86
Swedish-diverse	Faroese-leaky	55.36	44.06	46.04	30.69	46.04
	Faroese-nonleaky	40.15	29.34	22.84	18.06	22.84
	Faroese-all	40.76	29.93	23.82	18.59	23.

Table 5: Results of our experiment on bilingual transfer for Faroese.