Pull your treebank up by its own bootstraps

Ziqian Peng, Kim Gerdes, Kirian Guiller

Pull your treebank up by its own bootstraps

Ziqian Peng[1] Kim Gerdes[1] Kirian Guiller[2]

(1) Lisn (CNRS), Université Paris-Saclay, France
(2) Modyco (CNRS), Université Paris-Nanterre, France

ziqian.peng@universite-paris-saclay.fr
kim.gerdes@universite-paris-saclay.fr
kguiller@parisnanterre.fr

RÉSUMÉ

Remontez les bretelles à votre treebank

Nous analysons la performance de récents analyseurs syntaxiques neuronaux dans la tâche d'amorçage d'un treebank, c'est-à-dire l'entraînement et l'analyse itérative afin d'améliorer la vitesse et la qualité de l'analyse syntaxique humaine. En effectuant une recherche extensive et heuristiquement guidée dans la vaste grille d'options (analyseur syntaxique, plongement, configuration, époques, taille du batch, taille de l'ensemble d'entraînement, schéma d'annotation, langue, méthode d'évaluation...), nous déterminons les configurations d'analyseurs syntaxiques les plus performantes : UDify et Trankit se partagent le podium en fonction de la taille de l'ensemble d'entraînement. Nous montrons également comment ces résultats sont intégrés dans l'outil d'annotation ArboratorGrew, et nous proposons quelques mesures préliminaires qui permettent de prédire la qualité de l'analyse syntaxique pour une nouvelle langue.

ABSTRACT

We analyze the performance of recent neural syntactic parsers in the task of bootstrapping a treebank, i.e. training and analyzing iteratively in order to enhance speed and quality of the human syntactic analysis. By conducting an extensive and heuristically guided search in the vast grid of options (parser, embedding, configuration, epochs, batch size, size of training set, annotation scheme, language, evaluation method…), we determine the best performing parser configurations: UDify and Trankit share the podium depending on the size of the training set. We also show how these results are integrated into the annotation tool ArboratorGrew, and we propose some preliminary measures that allow predicting the quality of the parse for a new language.

MOTS-CLÉS : treebanks, annotation, analyseurs syntaxiques, réseaux neuronaux, amorçage, langues sous-ressourcées.

KEYWORDS: treebanks, annotation, syntactic parsers, neural networks, bootstrapping, under-resourced languages.

1 Introduction

Treebanks are steadily gaining importance as a tool for conducting research in syntax but their development is resource hungry in researcher's working hours as well as in the development and usage of recent neural network based tools. This is one of the reasons why the set of languages in Universal Dependencies (UD) is heavily biased towards well-resourced languages although an increasing number of, albeit often small, treebanks are developed for lower-resource languages (see for example the TowerParse project, Glavaš and Vulić 2021). This fact limits the scope of typological data-based studies on treebanks. In the context of the ANR project Autogramm (2022-2025), we develop a set of new treebanks for low-resource languages in the SUD annotation scheme (Gerdes et al. 2018, 2019, 2021). It is easier to annotate in SUD when no pre-established grammar exists that would provide a distinction between content and function words, which is commonly the case for less-resourced languages. Furthermore, SUD has been shown to be cognitively more relevant (Yan and Liu 2019), and parser performance improves on function-word-as-head annotation schemes (Rehbein et al. 2017)

We want to provide the usually less computer-inclined field linguists with state-of-the-art tools to develop high-quality treebanks and thus fill some gaps in treebank-based typological studies. More concretely, we want to answer the common questions of any syntactician wanting to start a new treebank: How many sentences do I have to annotate before it makes sense to train a first model? What parsing quality can I expect? How often should I retrain and reparse? What parser, embedding, and configuration should I use? How long does it take on my GPU? Can we make educated guesses on these questions based on raw or POS-tagged text?

2 Analysis and results

Although syntactic parsers are less relevant than they used to be for many NLP downstream tasks, these tools are still under very active development, in particular in a linguistic or low-resource perspective, and finding the parser best fitting for a given task is a quickly moving target. We chose 5 recent parsers: UDify (Kondratyuk and Straka 2019), Hopsparser (Grobol and Crabbé 2021), Trankit (Nguyen et al. 2021), Stanza (Qi et al 2020), BertForDeprel (Guiller 2020) and we tested their performance on 5 typologically diverse languages, English (en), French (fr), Chinese (zh), Japanese (ja) and Arabic (ar). We tested the parsers for six training sizes with the number of sentences [10, 30, 50, 100, 300, 500] during 100 epochs with 10-fold cross-validation, which gives us 5x5x6x10=1500 models to train and to evaluate. These numbers of sentences seem to us to be a reasonable grid for bootstrapping during the annotation process of a treebank for a new language.

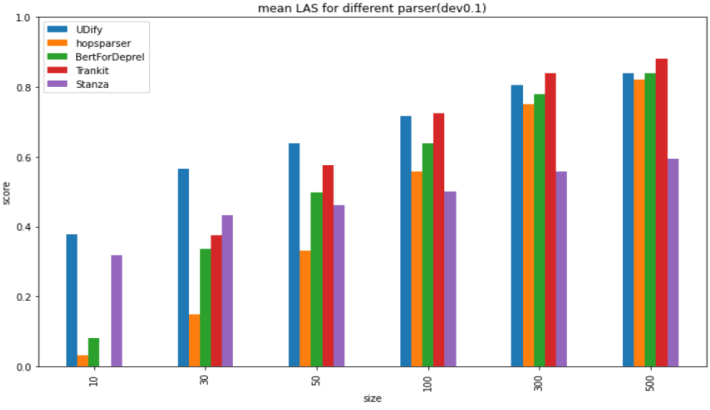


Figure 1: average parser performance for the 5 test languages: Training size vs. Labeled Attachment Score (LAS)

To do this, we randomly selected from SUD v2.10[1] 1500 sentences for each language, with 500 for training and 1000 sentences as test files, to be parsed by each trained parser and evaluated with the official UD evaluation script, so as to make the evaluation scores comparable.

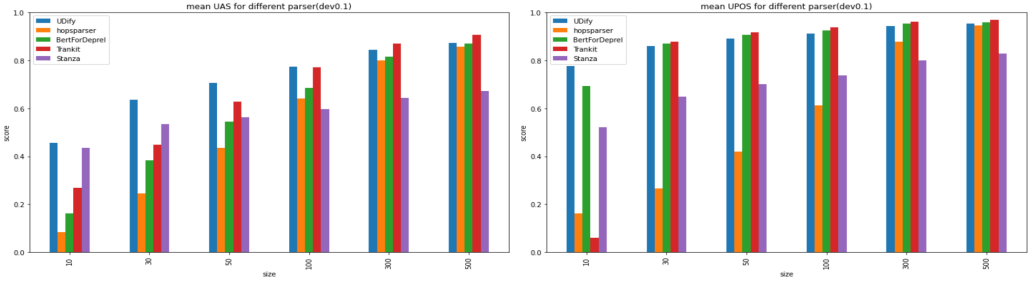


Figure 2: average parser performance for the 5 test languages: Training size vs. Unlabeled Attachment Score (UAS) at left and Training size vs. Universal POS tag (UPOS) at right

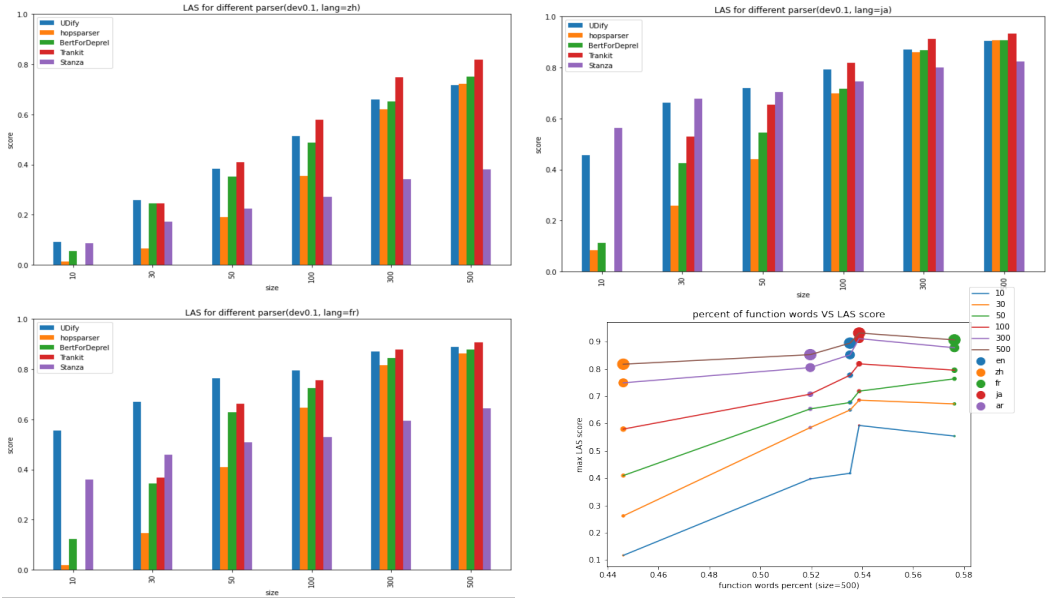


Figure 3: average parser performance for the 5 test languages: Training size vs. Labeled Attachment Score (LAS) for Chinese (first row at left, the worst performance as LAS), Japanese (first row at right, the best performance as LAS), French (second row at left) and the percent of functional words vs max LAS F1 score per language per size.

Before diving into the details, let us have a look at the average LAS (Labeled Attachment Score) results across our 5 languages: Figure 1 shows that UDify is clearly ahead when trained on a small training set. Starting with 100 sentences, Trankit takes the lead. These results are corroborated in the more detailed results below.

1 https://surfacesyntacticud.github.io/data/ In order to gather enough data, we had to merge several treebanks of each language.

2.1 Detailed parsing results for the 5 base languages

Just as for LAS, UDify starts the race of Unlabeled Attachment Scores (UAS) trained on very few sentences, but it is overtaken by Trankit only at 300 sentences. For precision during POS tagging, Trankit already takes the lead with 30 sentences to train on. When distinguishing the results by language, we first note that LAS takes more training data on Chinese than on other languages to reach comparable scores. Japanese and French, on the contrary, have above-average performance in LAS.

In the last graph of Figure 3 on the right, we ordered the languages by their percentage of function words, from 45% for Chinese (zh) to 57% for French (fr). As expected, we observe a general tendency of faster learning in languages with more function words, but the results for French are less good than for Japanese although it has more function words. Note that differences between the languages get less prominent the more training sentences we have.

2.2 Detailed parsing results for all available SUD treebanks

These first tests were based on the above-mentioned 5 languages. Based on these results, we repeated limited tests on the 69 languages of SUD 2.10 where 1500 sentences are available: We only tested on the two best-performing parsers Trankit and UDify, and we did not perform cross-validation.

When looking at Trankit's and UDify's performance per simple dependency relation (grouping subrelations, such as comp:ob under comp, see confusion matrices in the Annex), we see that the worst scores appear for the rare relations such as *orphan*, *reparandum,* and *list*) with a precision of 34%, 57% and 60% respectively (36%, 26%, and 72% respectively for UDify). Trankit's highest confusion rate is found for *udep* vs *orphan* with 18%. For UDify it is *reparandum* vs. *root* that causes a 34% confusion, pointing to a different tree spanning algorithm to create the trees.

For LAS, UAS, and POS tagging, we measure the average of both parsers. The results show very high discrepancies between the languages, ranging from 90% LAS for Greek to 22% for Coptic. The ancient languages are characterized by the fact that they are not easy to parse. It is not readily possible to determine the cause of these results, as it may be the genre of the texts, the languages themselves, or the incoherent annotation that the parser cannot pick up. It is noticeable that no language group stands out as being particularly easy to parse.

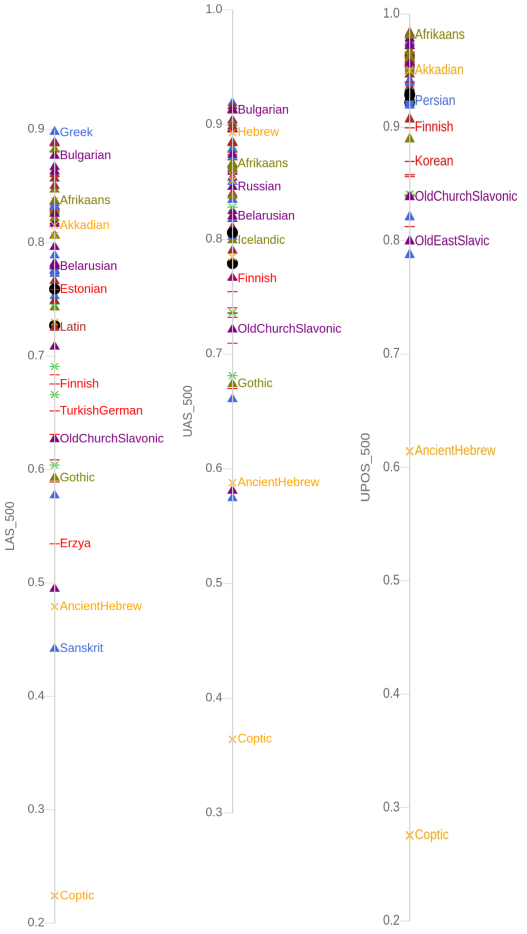


Figure 4: Average performance of Trankit and UDify measured by LAS (left), UAS (middle), UPOS (right) on dataset of size 500 for each of the 69 languages

Unsurprisingly, the LAS performance of the parsers is highly correlated with the POS accuracy. Put differently, as soon as we know the performance on POS tagging, we can predict reasonably well the LAS performance of the syntactic parser. Our data suggests that for 500 sentences in the language L, the LAS score can be computed by LAS(L)=1.55*POS-0.68.

The parser performance measured on the 69 languages confirms what we have observed before on our 5 test languages: Trankit needs at least 100 training sentences to catch up to Udify, but then delivers better parser results. See Figure A in Annex I for a graphical representation of these measures.

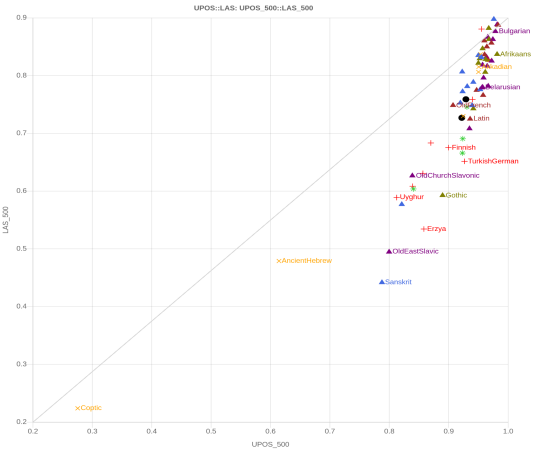


Figure 5: average performance of Trankit and UDIfy per language on dataset of size 500: Universal POS tag (UPOS) vs labeled Attachment Score (LAS)

3. Predicting parser performance

The observed significant differences in parser performance make it hard to give general predictions on the parser performance during treebank bootstrapping. Are there other measures that can be performed on raw or POS-tagged texts that can help us make better predictions? In this section we will show how the type/token ratio and the percentage of function words influences the parser performance, which allows us to make predictions of parser performance based on these measures. These findings are implemented as heuristics for automatically tweaking the parser parameters to optimize parser performance in the ArboratorGrew annotation tool (Guibon et al. 2020).

3.1 Can the type/token ratio predict parser performance?

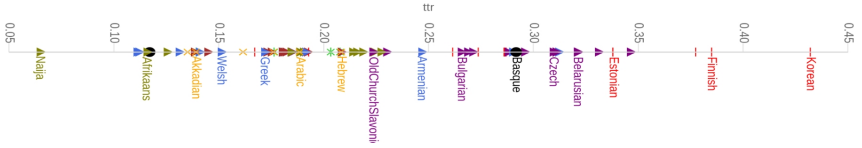


Figure 6: Type/Token ratio of the 69 languages, evaluated on the 1500 selected sentences.

The type/token ratio (TTR) is a measure of lexical richness of a text. As the TTR decreases for all languages with the size of the text, we need to measure it on texts of approximately the same length in order to make it comparable. The plot of Figure 6 shows large differences between the languages, the lead being taken by agglutinating languages followed by Slavic languages, and Korean being the “richest” language with a TTR of 43%. The large TTR difference between structurally similar languages such as Korean and Japanese (at only 19% not shown above) can be explained by different word segmentation rules underlying the treebanks: Japanese is separating the verbal and nominal suffixes, resulting in many equal functional tokens, and Korean considers the suffixes as

part of the word, resulting in many unique tokens.

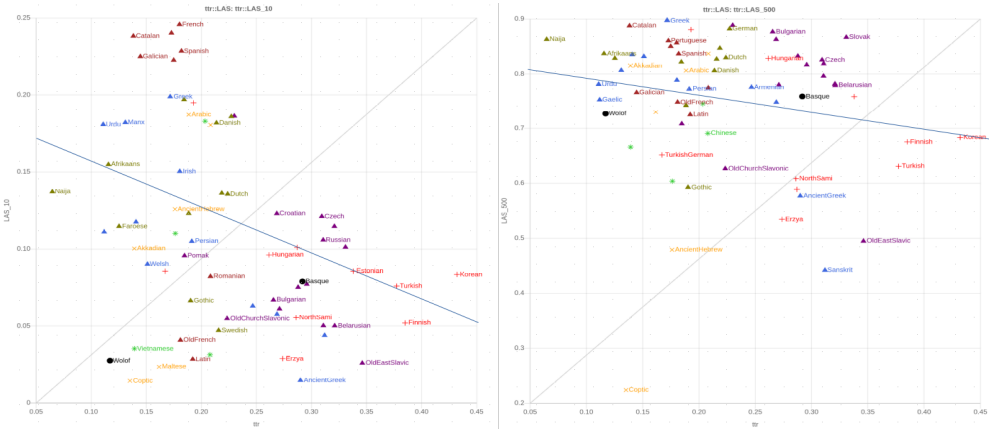


Figure 7: Type/Token Ration (TTR) vs LAS on dataset of size 10 (left) and that of size 500 (right), with blue lines illustrating correlation between TTR and LAS (cf. Annex VI)

As expected, we observe a negative correlation between TTR and LAS: The richer the language the harder it is to parse. Also, the Spearman correlation coefficient decreases between scores for training on 10 and on 500 sentences, respectively -0.33 and -0.17, indicating that the measure becomes less relevant with larger training sets.

3.2 Can POS tags predict parser performance?

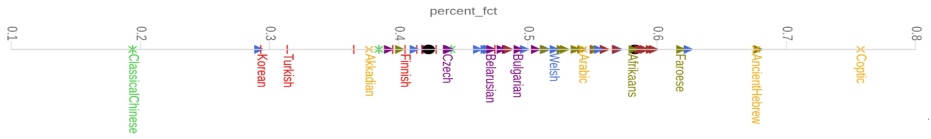


Figure 8: overview of the percentage of function words across languages, evaluated on the 1500 selected sentences.

We have shown above that POS tagging performance is a very good indicator of LAS performance. But can the distribution of POS themselves be a predictor? Our hypothesis is that the lexical vs. function word distinction allows us to make predictions: The more function words, the easier. Taking nouns, verbs, adjectives, and adverbs as lexical categories, we first observe a distribution ranging from 20% function words in classical Chinese to 76% in Coptic. Plotting these measures against the LAS score, we observe the expected positive correlation. The two languages with the highest percentage of function words, Coptic and Ancient Hebrew, are outliers of the general tendency. The Spearman correlation coefficient is 0.4453 for 500 sentences.

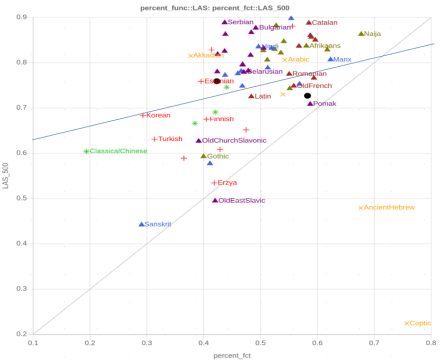


Figure 9: Percentage of function words of the 69 languages computed with selected sentences VS LAS F1 score on dataset of size 500

3.3 Parser performance and language structure

Two other interesting results are the measures of language directionality and tree height: When measuring tree directionality (the average dependency length, counting leftward relations negatively), we observe that this measure has little influence on the parser results (Spearman correlation 0.3230 with p-value 7.6% > 5% to accept the null hypothesis that the observed correlation is due to chance). Inversely, the tree height has a very profound influence on the parser: the higher the tree, the better the score (Spearman 0.5126). This latter correlation may be another explanation for the better parser performance of SUD vs UD: SUD's function word centric approach simply results in higher trees. The integration of the parser results into the typometrics platform https://typometrics.elizia.net allows for further study of the correlation between various treebank measures and treebank performance.

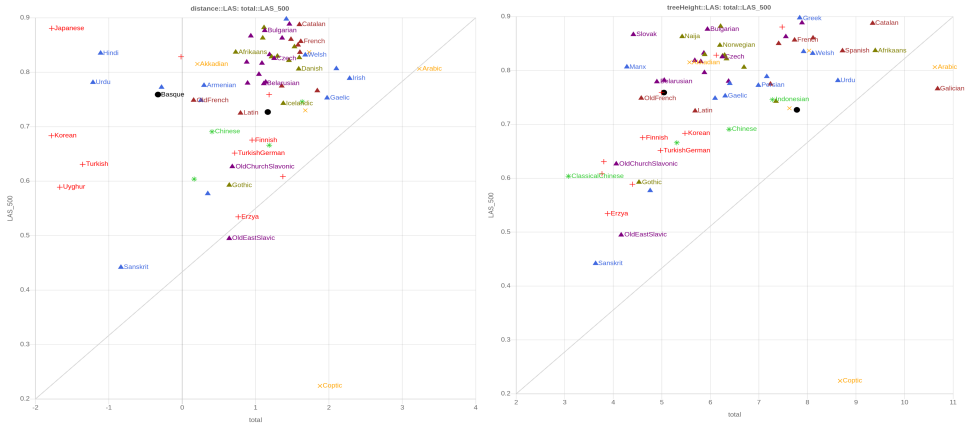


Figure 10: Distance computed on SUD2.8 VS LAS F1 score on dataset of size 500 (left) and treeheight computed on SUD2.8 VS LAS F1 score on dataset of size 500 (right)

4. Implementation in ArboratorGrew

ArboratorGrew is a new treebank annotation tool that integrates Grew's graph search and rewrite features into Arborator's collaborative online annotation platform. The new train-and-parse option makes it possible to use any of the five parsers to train a model on some samples, and obtain the parse results on other samples. The parser operates on a separate server equipped with a high-performance graphics card.[2] The interface proposes simple options and makes predictions on the required time to train and parse based on a logical regression, see the Annex for a screenshot and for the time regression lines.

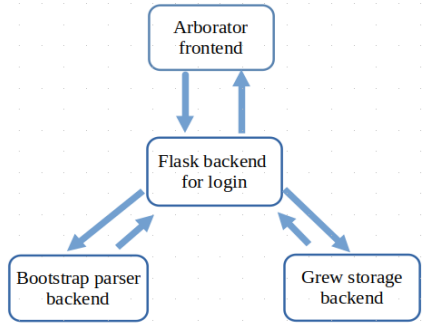


Figure 11: architecture of frontend and backend.

2 The bootstrapping backend relies on a single Nvidia RTX A6000 card with 48Gb of RAM.

5. Conclusion

The influence of annotating pre-analyzed text has been discussed in Fort & Sagot 2010, and we should be aware that the syntactician is less likely to detect new peculiar constructions when presented with a reasonably well pre-annotated text. On the other hand, the lower diversity of the pre-analyzed treebank annotation naturally results in better parser performance.

The rather quick increase in LAS with the size of the training set suggests a very early and regular bootstrapping approach, possibly best starting with 30 sentences. When exactly this makes sense heavily depends on the language, and, of course, on the time measures on parse corrections compared to an annotation from scratch. With such a measure, which remains to be done on a variety of annotators' profiles, it would be possible to answer for example whether a pre-annotation with only 50% LAS is still useful or not.

We also see the need to improve the ArboratorGrew tool: The "diff" mode showing the difference between two trees should show the certainty of proposed relations, so as to allow the annotator to see directly the problematic relations that require scrutiny. Also, a single manual correction should optionally trigger a recomputation of the minimum spanning tree so that the most likely structure, given the new relation, can be proposed directly without further manual intervention. This would significantly reduce the correction time spent on faulty parse results.

Acknowledgements

We would like to thank our anonymous reviewers for their interesting remarks and questions. Loïc Grobol helped with the implementation of the Hops parser and pointed us to Trankit. Laurent Pointal helped us to develop a secure parser backend, and the intensive parser training was done on both the Lisn and the Lab-ia clusters at the University Paris-Saclay.

References

Fort, Karën, and Benoît Sagot. "Influence of pre-annotation on POS-tagged corpus development." In The fourth ACL linguistic annotation workshop, pp. 56-63. 2010.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, Guy Perrier. Starting a new treebank? Go SUD! Theoretical and practical benefits of the Surface-Syntactic distributional approach in DepLing 2021.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, Guy Perrier. Improving Surface-syntactic Universal Dependencies (SUD): surface-syntactic relations and deep syntactic features in TLT 2019.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, Guy Perrier. SUD or Surface-Syntactic Universal Dependencies: An annotation scheme near-isomorphic to UD in UDW 2018.

Goran Glavaš and Ivan Vulić, “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, 2021, pp. 4878–4888.

Loïc Grobol and Benoît Crabbé, “Analyse en dépendances du français avec des plongements contextualisés,” in TALN/RECITAL 2021, 2021.

Guibon, Gaël, Marine Courtin, Kim Gerdes, and Bruno Guillaume. "When collaborative treebank curation meets graph grammars." In LREC 2020 -- 12th Language Resources and Evaluation Conference. 2020.

Kirian Guiller. "Analyse syntaxique automatique du pidgin-créole du Nigeria à l’aide d’un transformer (BERT): Méthodes et Résultats." Mémoire de Master, Sorbonne Nouvelle (2020).

Dan Kondratyuk and Milan Straka, “75 languages, 1 model: Parsing universal dependencies universally,” arXiv preprint arXiv:1904.02099, 2019.

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen, “Trankit: A light-weight transformer-based toolkit for multilingual natural language processing,” arXiv preprint arXiv:2101.03289, 2021.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D Manning, “Stanza: A python natural language processing toolkit for many human languages,” arXiv preprint arXiv:2003.07082, 2020.

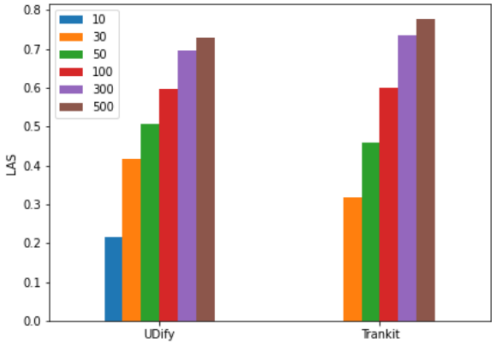
Rehbein, Ines, Julius Steen, Bich-Ngoc Do, and Anette Frank. "Universal Dependencies are hard to parse–or are they?." In Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017), pp. 218-228. 2017.

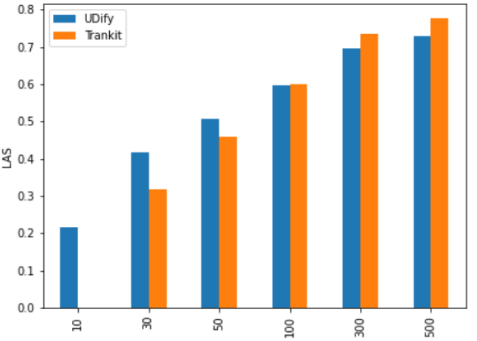
Yan, Jianwei, and Haitao Liu. Which annotation scheme is more expedient to measure syntactic difficulty and cognitive demand?. Presented at Quasy, SyntaxFest 2019.

Annex

I. Statistic of LAS f1 score for Trankit and UDify

Figure A: Performance of UDify and Trankit for the 69 available languages in SUD2.10 : parser vs Labeled Attachment Score (LAS) at left and Training size vs LAS at right





Statistic of LAS f1 score for Trankit (left) and UDify (right) on 69 languages

	10	30	50	100	300	500
count	69.0	69.000000	69.000000	69.000000	69.000000	69.000000
mean	0.0	0.318216	0.457290	0.600810	0.735069	0.776311
std	0.0	0.083658	0.120591	0.131782	0.125436	0.117720
min	0.0	0.154317	0.170476	0.195329	0.219645	0.219452
25%	0.0	0.253727	0.377353	0.528295	0.672990	0.731561
50%	0.0	0.327911	0.468934	0.639530	0.776533	0.806653
75%	0.0	0.381771	0.556436	0.689709	0.820712	0.850298
max	0.0	0.475108	0.653467	0.788150	0.885604	0.914372

	10	30	50	100	300	500
count	69.000000	69.000000	69.000000	69.000000	69.000000	69.000000
mean	0.220727	0.416438	0.508161	0.595730	0.694433	0.727304
std	0.127837	0.153908	0.156169	0.153074	0.140600	0.134007
min	0.028970	0.122755	0.166573	0.210142	0.223761	0.228609
25%	0.115289	0.303778	0.388442	0.508322	0.630197	0.670072
50%	0.202076	0.409790	0.530283	0.636877	0.741292	0.767542
75%	0.309760	0.544047	0.648022	0.714934	0.793280	0.816323
max	0.491621	0.680553	0.761930	0.805829	0.861106	0.881114

Languages with LAS less than 0.5 when the dataset contains 500 sentences: Trankit is more universal than UDify so that only 2 languages got LAS less than 0.5 with dataset of size 500.

```
las_trankit[las_trankit['500'] < 0.5]
```

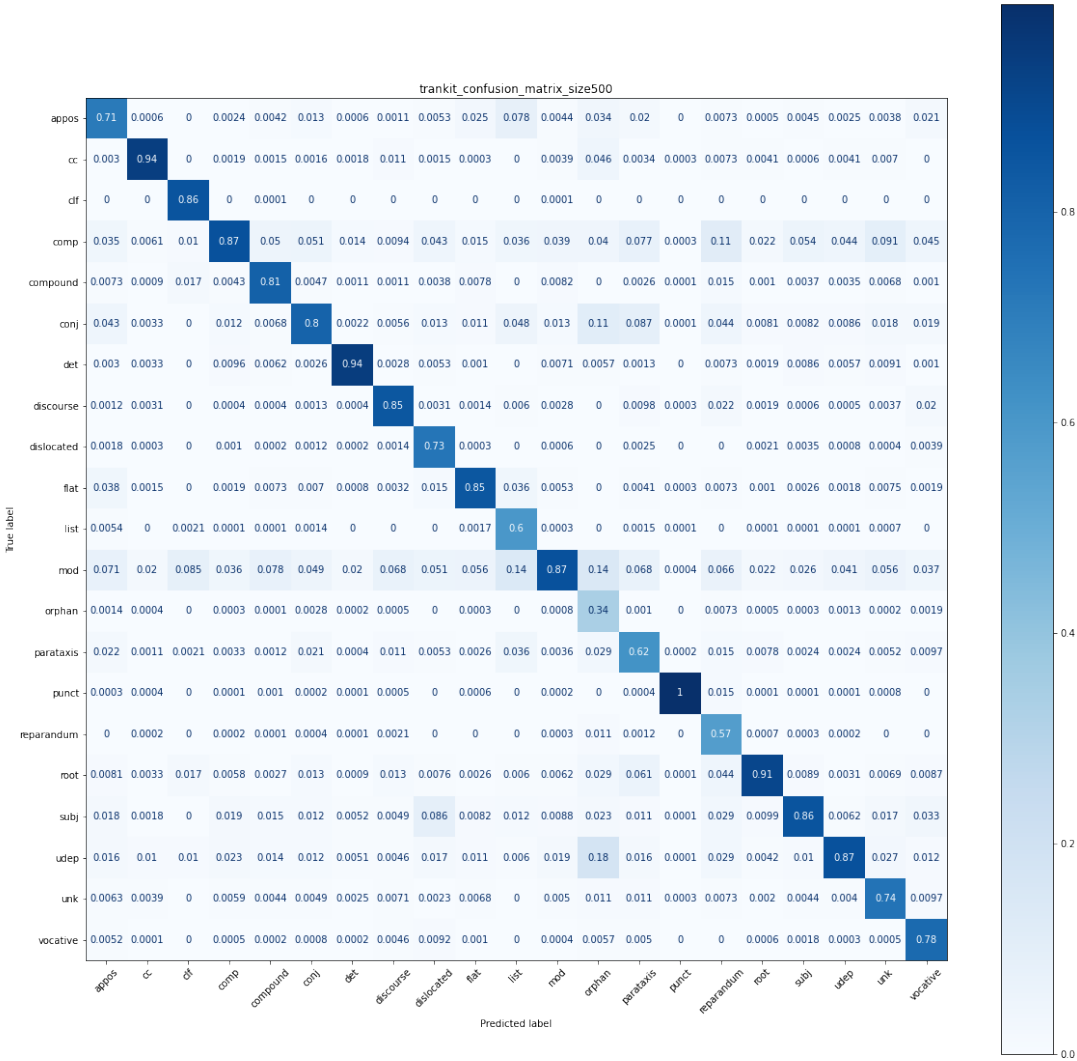
	10	30	50	100	300	500
cop	0.0	0.154317	0.170476	0.195329	0.219645	0.219452
sa	0.0	0.159379	0.200846	0.252045	0.390550	0.486037

```
las_udif[las_udif['500'] < 0.5]
```

	10	30	50	100	300	500
cop	0.028970	0.168629	0.184672	0.210142	0.223761	0.228609
grc	0.029599	0.188796	0.200606	0.304921	0.429826	0.477139
hbo	0.251773	0.295608	0.332957	0.364424	0.393916	0.407937
orv	0.051763	0.166951	0.196107	0.275949	0.391675	0.433659
sa	0.087870	0.144147	0.166573	0.226375	0.329055	0.397743
ug	0.202076	0.218613	0.299190	0.347368	0.432838	0.471903

II. Confusion matrix for Precision

II.1 Trankit:



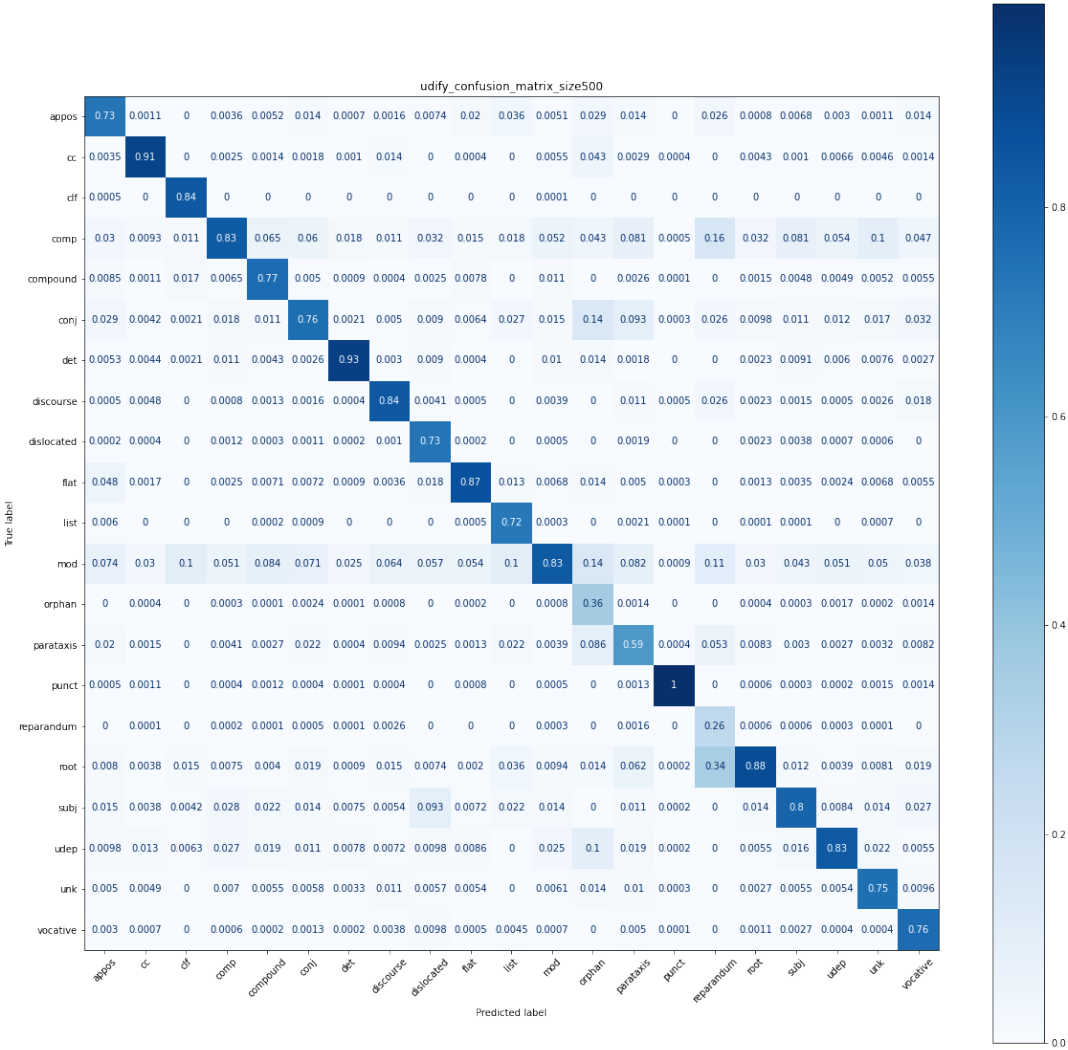
For Trankit std < 0.08 (left), std > 0.15: (right)

	cc	clf	det	punct	udep
std	0.034557	0.07126	0.032405	0.004537	0.05992

	appos	conj	list	parataxis	reparandum
std	0.174365	0.217287	0.166138	0.157149	0.158298

This table shows the standard variation over the different training sizes from 10 to 500 for various relations. E.g. The lowest standard variation is for *cc* which indicates that the analysis does not improve significantly with a bigger training set. On the contrary, rare relations such as appos, conj, list, parataxis, and referendum are still varying a lot and can be expected to improve with a larger training set. Check also the last column of F1 score for Trankit and UDify in Annex III.

II.2 UDify



Amount of different Deprel in the 69 languages:

	appos	cc	clf	comp	compound	conj	det	discourse	dislocated	flat	goeswith	list	mod	orphan	parataxis	punct	reparandum	root	subj	udep	unk	vocative
amount	9152	45089	447	320594	21454	51185	81468	6282	2139	15809	35	393	235847	768	8729	151395	368	69000	89670	123638	13101	1517

We exclude *goeswith* from the confusion matrix since there are only 35 occurrences of this relation.

III. F1 score for Trankit(left) and UDify(right) by size

	10	30	50	100	300	500	500-50
appos	0.0	0.162	0.207	0.390	0.560	0.632	0.425
cc	0.0	0.847	0.872	0.899	0.931	0.938	0.066
clf	0.0	0.611	0.741	0.822	0.878	0.892	0.150
comp	0.0	0.682	0.752	0.816	0.874	0.891	0.139
compound	0.0	0.539	0.612	0.700	0.772	0.790	0.178
conj	0.0	0.228	0.357	0.536	0.736	0.792	0.435
det	0.0	0.818	0.851	0.889	0.919	0.928	0.077
discourse	0.0	0.461	0.534	0.646	0.769	0.805	0.270
dislocated	0.0	0.344	0.364	0.417	0.523	0.556	0.192
flat	0.0	0.375	0.510	0.644	0.774	0.811	0.301
goeswith	0.0	0.000	0.000	0.000	0.000	0.089	0.000
list	0.0	0.019	0.034	0.129	0.311	0.354	0.320
mod	0.0	0.650	0.714	0.782	0.849	0.871	0.157
orphan	0.0	0.009	0.017	0.043	0.072	0.125	0.108
parataxis	0.0	0.181	0.259	0.341	0.478	0.541	0.283
punct	0.0	0.989	0.993	0.996	0.997	0.998	0.005
reparandum	0.0	0.116	0.153	0.198	0.234	0.309	0.156
root	0.0	0.700	0.780	0.841	0.893	0.909	0.129
subj	0.0	0.573	0.678	0.759	0.837	0.862	0.184
udep	0.0	0.737	0.779	0.814	0.856	0.871	0.092
unk	0.0	0.273	0.378	0.496	0.621	0.663	0.286
vocative	0.0	0.191	0.295	0.365	0.521	0.627	0.333

	10	30	50	100	300	500	500-50
appos	0.050	0.177	0.276	0.387	0.536	0.579	0.303
cc	0.643	0.824	0.844	0.873	0.907	0.915	0.071
clf	0.423	0.671	0.780	0.817	0.855	0.869	0.089
comp	0.524	0.667	0.723	0.778	0.838	0.856	0.133
compound	0.198	0.462	0.536	0.617	0.704	0.730	0.194
conj	0.085	0.255	0.379	0.523	0.680	0.733	0.354
det	0.702	0.798	0.833	0.868	0.904	0.913	0.080
discourse	0.290	0.420	0.497	0.580	0.704	0.743	0.245
dislocated	0.074	0.219	0.249	0.389	0.494	0.532	0.283
flat	0.244	0.435	0.549	0.645	0.765	0.793	0.244
goeswith	0.000	0.000	0.000	0.000	0.000	0.000	0.000
list	0.021	0.064	0.052	0.156	0.288	0.519	0.468
mod	0.425	0.588	0.665	0.730	0.803	0.828	0.163
orphan	0.000	0.009	0.020	0.019	0.054	0.060	0.040
parataxis	0.053	0.194	0.251	0.320	0.443	0.494	0.243
punct	0.844	0.981	0.988	0.992	0.995	0.996	0.008
reparandum	0.005	0.040	0.034	0.033	0.050	0.049	0.016
root	0.492	0.686	0.744	0.803	0.862	0.880	0.136
subj	0.305	0.501	0.598	0.679	0.770	0.800	0.203
udep	0.568	0.696	0.733	0.776	0.822	0.839	0.106
unk	0.066	0.236	0.332	0.438	0.559	0.611	0.280
vocative	0.177	0.251	0.322	0.349	0.433	0.497	0.175

Note that we cannot train the Trankit pipeline for our dataset of language ga (Irish) with only 10 sentences. The last column of both tables reporte the improvement of F1 score with the augmentation of data size from 50 to 500. The score for conj, appos and list have been improved more than 30% with both parsers.

IV. List of the 69 languages :

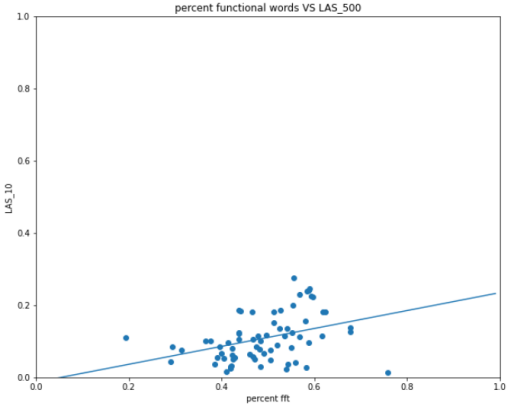
Afrikaans, Akkadian, AncientGreek, AncientHebrew, Arabic, Armenian, Basque, Belarusian, Bulgarian, Catalan, Chinese, ClassicalChinese, Coptic, Croatian, Czech, Danish, Dutch, English, Erzya, Estonian, Faroese, Finnish, French, Gaelic, Galician, German, Gothic, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Irish, Italian, Japanese, Korean, Latin, Latvian, Lithuanian, Maltese, Manx, Naija, NorthSami, Norwegian, OldChurchSlavonic, OldEastSlavic, OldFrench, Persian, Polish, Pomak, Portuguese, Romanian, Russian, Sanskrit, Serbian, Slovak, Slovenian, Spanish, Swedish, Turkish, TurkishGerman, Ukrainian, Urdu, Uyghur, Vietnamese, Welsh, WesternArmenian, Wolof

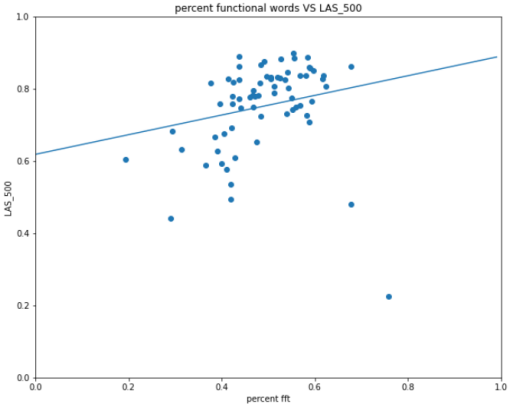
V. Parser configuration:

For all parsers 10% for the dev set after comparison between 10%, 20% and 30%.

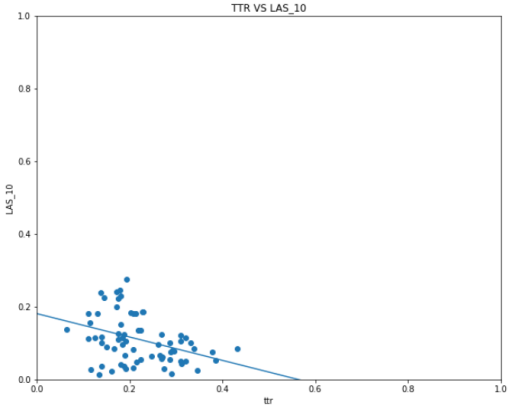
VI. Correlation between metrics

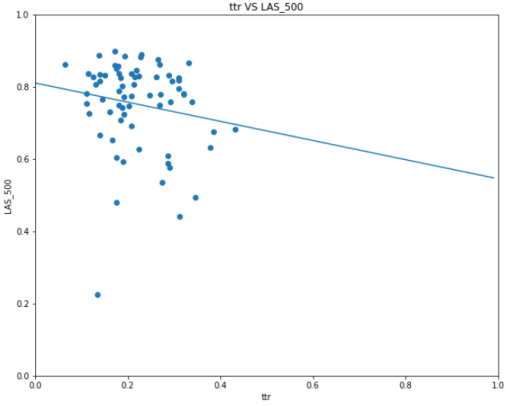
VI.1 Percent of functional words VS LAS for dataset of size 10 (left) and 500 (right)



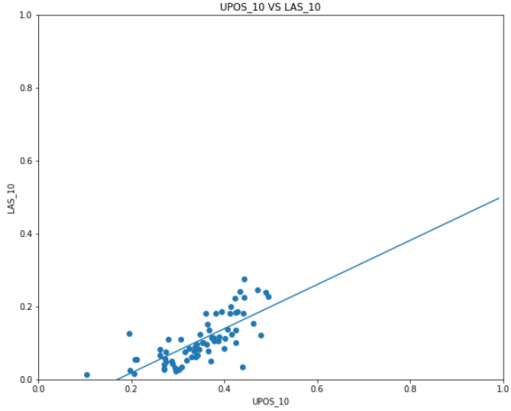


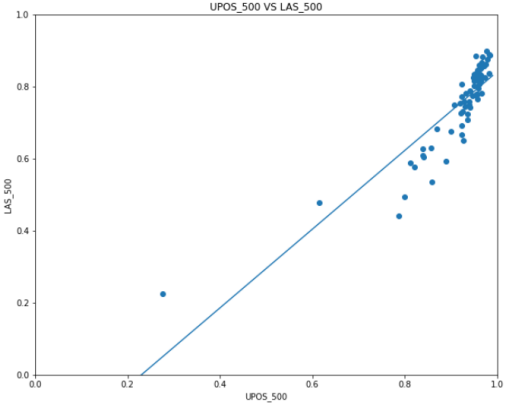
VI.2 Type-token ratio VS LAS for dataset of size 10 (left) and 500 (right)





VI.3 UPOS VS LAS for dataset of size 10 (left) and 500 (right)

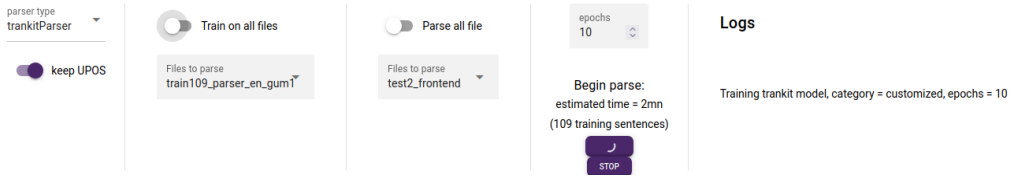




VII. The ArboratorGrew implementation

Screenshot of AboratorGrew's parse options panel:

Users can first choose the gold files as training set, the files to parse, then the parser type such as trankitParser for Trankit and the number of epochs. The keep UPOS option indicates whether the UPOS in selected files to parse need to be kept. If we click the 'begin parse' button, a log message appears to show the current progress, such as data preparation, training and parsing.



A rough time estimation for each parser:

Empirically, the training and parsing time increases faster than logarithm but slower than a simple line regression, so the logical regression is computed with the following parameters:

$$ftime = A*log(x + 1) + B*x + C.$$

Note that the effective consumed time may be less than estimated with larger training data.