A Modest Pareto Optimisation Analysis of Dependency Parsers in 2021

Mark Anderson
Universidade da Coruña, CITIC
Department of CS & IT
m.anderson@udc

Carlos Gómez-Rodríguez
Universidade da Coruña, CITIC
Department of CS & IT
carlos.gomez@udc.es

Abstract

We evaluate three leading dependency parser systems from different paradigms on a small yet diverse subset of languages in terms of their accuracy-efficiency Pareto front. As we are interested in efficiency, we evaluate core parsers without pretrained language models (as these are typically huge networks and would constitute most of the compute time) or other augmentations that can be transversally applied to any of them. Biaffine parsing emerges as a well-balanced default choice, with sequence-labelling parsing being preferable if inference speed (but not training energy cost) is the priority.

1 Introduction

The inefficiency of modern NLP systems has recently come under scrutiny, especially regarding their large energy consumption (Strubell et al., 2019). This hasn't started a revolution, but there is some NLP work where efficiency is considered. Zhang and Duh (2020) studied different settings for neural machine translation systems, evaluating not only accuracy but also certain costs such as inference time, training time, and model size. Zhou et al. (2021) analysed the fine-tuning and inference time for pretrained LMs, and estimated the cost of pretraining. Jacobsen et al. (2021) presented a Pareto optimisation analysis for POS taggers, considering accuracy and model size.

In parsing in particular, Strzyz et al. (2019) evaluated dependency parsing as sequence labelling specifically to increase inference efficiency and also undertook a Pareto optimisation analysis. Others used model compression via distillation to increase inference speed of neural parsers with a mixed bag of results (Dehouck et al., 2020; Anderson and Gómez-Rodríguez, 2020a). Dehouck et al. (2020) also took into consideration the training energy costs of distilling models, which highlighted the high energy cost of this technique.

We present a Pareto optimisation analysis on modern dependency parsing systems. We cover three systems which are broadly representative of current approaches. We analyse their efficiency with respect to inference speed and also their training cost, measured in energy consumption.

Contribution: A simple, modest analysis on the merits of different parser systems that cover three current paradigms. Our goal is not to provide surprising results, but a realistic snapshot of the current state of affairs of a representative sample of modern parsing systems on linguistically diverse data. This analysis runs the systems in a consistent way with respect to software, hardware, and network settings. We also offer a brief overview of self-reported performance on PTB for systems that have a published speed. We add to this measurements for a subset of these systems which we ran locally for a more consistent comparison, i.e. something of a reproducibility effort.

Disclaimer We make a practical comparison for practitioners, so we focus on publicly available systems on typical hardware that doesn't require a huge budget. We are not making general claims that technique X is always more efficient than technique Y in the abstract or that this will hold in any hardware. Also, the extent to which an implementation has been engineered will impact performance, so we have referenced the original repositories used.[1]

2 PTB performance

For historical reasons, it is common practice for parsers to report performance results on the English Penn Treebank (PTB) (Marcus and Marcinkiewicz, 1993). While such results at best provide a partial picture on a single language, they are by far the

[1]The moderately edited code is available at http://www.grupolys.org/software/iwpt2021/parsers-code.zip.

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 119–130
Bangkok, Thailand (online), August 6, 2021.

	speed (sent/s) GPU	CPU	UAS	LAS
HPSG (Zhou and Zhao, 2019)	159*	-	96.09*	94.68*
Biaffine w CRF (Zhang et al., 2020a)	400*		96.14*	94.49*
Pointer-LR (Fernández-González and Gómez-Rodríguez, 2019)	23*	-	96.04*	94.43*
GNN (Ji et al., 2019)	416*	-	95.97*	94.31*
Pointer-TD (Ma et al., 2018)	10.2†	-	95.87*	94.19*
Biaffine (Dozat and Manning, 2017)	411*	-	95.74*	94.08*
Distilled-Ensemble (Kuncoro et al., 2016)	-	20*	94.26*	92.06*
BIST - Transition (Kiperwasser and Goldberg, 2016)	-	76±1‡	93.9*	91.9*
SeqLab (Strzyz et al., 2019)	648±20*	101±2*	93.67*	91.72*
BIST - Graph (Kiperwasser and Goldberg, 2016)	-	80±0‡	93.1*	91.0*
CM (Chen and Manning, 2014)	-	654*	91.80*	89.60*
Pointer-LR	95±1	8±0	96.02	94.47
Biaffine (PyTorch)	1003±3	53±0	95.74	94.07
UUParser (Smith et al., 2018)	-	42±1	94.63	92.77
Distilled-Biaffine (Anderson and Gómez-Rodríguez, 2020a)	1153±3	96±0	94.59	92.64
SeqLab	1064±13	99±1	93.46	91.49
MaltParser 1.9.2 w/ Stack lazy (Nivre et al., 2007)	-	473±11	89.29	86.95

Table 1: Performance for current leading parsers for the English PTB with POS tags predicted from the Stanford POS tagger. ∗ denotes values taken from the original paper, † from Fernández-González and Gómez-Rodríguez (2019), and ‡ from Strzyz et al. (2019). Values with no superscript are from running the models on our system locally (speeds averaged over 5 runs) and with a batch size of 256 (excluding UUParser which doesn't support batching) with GloVe 100 dimension embeddings. Table is extended from one in Anderson and Gómez-Rodríguez (2020a).

most comprehensive source of results provided in the literature under a consistent context (at least in terms of data and splits, although not hardware), so they are useful to see high-level trends and as a starting point to choose parsers for our experiment.

In Table 1 we report performance of modern parsing systems for which speeds have been reported. We couldn't find a reported speed of Clark et al. (2018) which currently has the highest reported performance on PTB (UAS 96.61 and LAS 95.02) when not using BERT. However, its main contribution is semi-supervised augmentations that could be utilised by any parsing system, with their core parser being the Biaffine parser. Zhou and Zhao (2019)'s system leverages constituency and dependency parsing and when not using training data with both constituency and dependency annotations (often not available) the system achieves UAS 95.82 LAS 94.43 (i.e. very similar in LAS to the other top-performing sytems). Zhang et al. (2020a) use a Biaffine parser but with a moderate beam search, which is obviously less efficient than the original. It results in a small increase in performance. Ji et al. (2019) use graph neural networks to learn enriched high-order information from partial parses. It again only gains small increases over Biaffine, but is more computationally complex and code is not available.

We report results for UUParser of Smith et al. (2018) that we ran locally (refreshingly the original paper didn't use PTB). While the results show a reasonable speed-accuracy trade-off, we opted not to use this for the current analysis as the original code is implemented in DyNet which doesn't properly support CUDA, and is a different framework from that of the other parsers we opted to choose.

Based on this, we opted to use the basic Biaffine parser to represent graph-based parsers, the Pointer-LR network as the representative of transition-based algorithms,[2] and the sequence-labelling parser to represent SL systems. They all have the added benefit of working under the same software and having code available.

Note that, as we make emphasis on efficiency, we focus on reasonably bare-bones versions of the parsers. The impact of pretrained language models, or other augmentations that are transversal to the parsing system, is outside the scope of this paper.

3 Pareto optimisation analysis

Here we detail the parsing systems, the data we used, and how model structures were altered.

3.1 Parsers

All the parsers use BiLSTMs, but have additional structures which set them apart from one another

[2]Some might argue that it isn't a clear cut case of a transition-based parser, but it transitions from state to state like more traditonal algorithms.

and use one of three paradigms broadly speaking: one is a transition-based parser, one is a sequence-labelling parser, and the last is a graph-based parser. For space reasons, we only very briefly outline them here, but give more details in Appendix A.

Left-to-right pointer network (L2R). One of the current top-performing parsers on PTB, it uses a left-to-right transition-based algorithm that builds a number of attachments equal to sentence length using a pointer network (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019).[3]

Deep biaffine (BIAFFINE) (Dozat and Manning, 2017) is an edge-factored graph-based parser that produces a matrix of scores giving a probability distribution on arcs, where the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) is then applied to obtain a tree.

Sequence labelling parser (SEQLAB) encodes trees as a sequence of labels, so that a direct one-to-one prediction can be made for each token in a sentence (Spoustová and Spousta, 2010; Li et al., 2018b; Strzyz et al., 2019).[4] We implement it using the Biaffine system described above (for uniformity) editing it to be a sequence-labelling system.

3.2 Data

In our choice of treebanks, we balance three factors: the need to use a small number of treebanks (as our detailed Pareto analysis implies training a large number of models per treebank), linguistic diversity and treebank quality. This leads us to choose 4 high-quality (manually annotated or corrected, and relatively large) treebanks covering 3 different language families and 4 subfamilies: UD-Hindi-HDTB, UD-Polish-PDB, UD-Korean-Kaist and the Chinese Penn Treebank. More details of each treebank, justifying their diversity and adequacy for the analysis are given in Appendix C.

3.3 Methodology

We vary the size of the BiLSTM component of the networks by their number of layers and nodes. Each parser has randomly-initialised character embeddings and pretrained word embeddings as only inputs. We use pretrained FastText embeddings (Grave et al., 2018). Except for Chinese, as the FastText embeddings are in the traditional script, so we use the embeddings from Li et al. (2018a).[5] The embeddings are reduced to 100 dimensions using PCA. The structure of the networks are very similar. The L2R system uses a biaffine transformation to score the transitions at each step similar to the BIAFFINE parser, and we use the same sizes for the layers. The SEQLAB system is altered from the BIAFFINE implementation and is exactly the same except the layers needed for the biaffine transformation are replaced by two MLPs which predict the labels for each token. The only major difference in the networks is that L2R uses a CNN to create the character embeddings and the other two use BiLSTMs. We didn't change this in order to avoid modifications to the systems. The network hyperparameters are shown in Table 2 in Appendix B. Models were trained on GPU, but we report the energy used by both the GPU and CPU.

We could have altered other aspects of the network, but the main computational cost comes from the BiLSTM layer. The other main contender to alter would be the embedding layers. For example, we could have altered the size of the character BiLSTM/CNN, but certain experiments show that it has a limited impact on accuracy (Smith et al., 2018; Anderson and Gómez-Rodríguez, 2020b).

We measured the speed of each system on each treebank by running them 5 times using a single CPU core, both for speeds measured running on GPU and CPU, so that we get a reasonably accurate measure of the speed for each treebank. We then report macro averaged speeds across treebanks.

We use the `energyusage` package for measuring training energy.[6] It measures the power usage of the GPU and CPU while a process is running (having taken a measure of the background usage). We minimised the use of the system when training these models to obtain accurate measurements, but they aren't overly precise. This isn't a major issue as the measurements are over long periods of time and so unless there were massive fluctuations when training a given model, comparison is fine. We use joules (or kJ and MJ) as they are the SI units for energy (BIPM, 2019) and, unlike carbon emissions, they are independent of external factors like regional electricity generation grids.

Hardware: Intel Core i7-7700 and Nvidia GeForce GTX 1080.

Software: Python 3.7.0, PyTorch 1.0.0, and CUDA 8.0.

[3] https://github.com/danifg/SyntacticPointer.

[4] We use refactored encoding/decoding functions from https://github.com/mstrise/dep2label.

[5] https://jima.me/open/cwv/

[6] https://pypi.org/project/energyusage

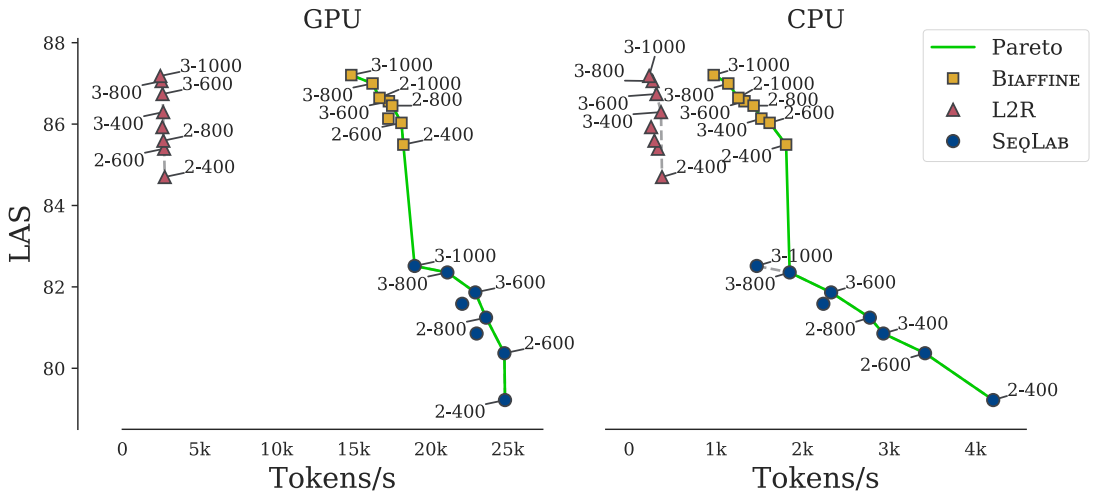

Figure 1: Pareto fronts for L2R, BIAFFINE, and SEQLAB for the development data.

3.4 Pareto fronts: inference speed

Figure 1 shows LAS versus parsing speed for the development data (we also present the same for the test data in Figure 7 in the Appendix that echoes the visualisation seen here). The individual Pareto front for each parser is shown (light grey, dashed) As expected, models with larger networks are more accurate but slower. More interestingly, the overall Pareto front is exclusively constructed of BIAFFINE and SEQLAB systems. While L2R does achieve similar accuracy scores as BIAFFINE, it is considerably slower. SEQLAB is the fastest option by a clear margin (especially smaller networks on CPU). So the practical advice to draw from this aspect or the Pareto optimisation would be to use BIAFFINE if accuracy is the main concern, or SEQLAB if inference time is important.

3.5 Pareto fronts: training energy

Figure 2 shows LAS against the average energy (across treebanks) consumed during training (in training, we always use the GPU). There is no clear link between the energy consumed and the accuracy of a system. However, this visualisation highlights that SEQLAB is nowhere near optimal with respect to training efficiency.

The amount of energy consumed during training is basically dependent on the time it takes each system to converge as can be seen in Figure 3. In this figure, we show individual models (i.e. not averaged over treebanks). The relation for BIAFFINE and SEQLAB is very clearly linear between energy and training time, suggesting that there is nothing intrinsically more energy consuming between these systems beyond convergence time. For L2R, this relation seems to hold broadly, but is less clear. It appears that L2R is more sensitive to the nature of

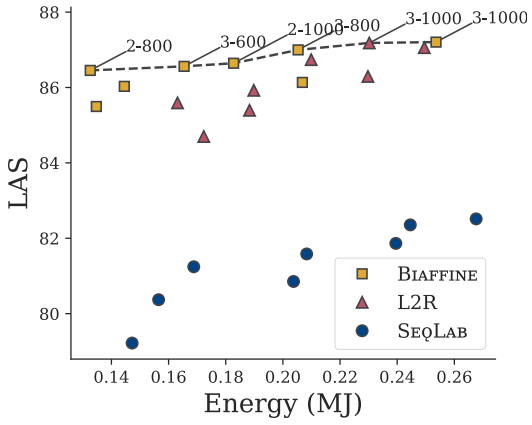

Figure 2: Pareto fronts for L2R, BIAFFINE, and SEQLAB for training energy.

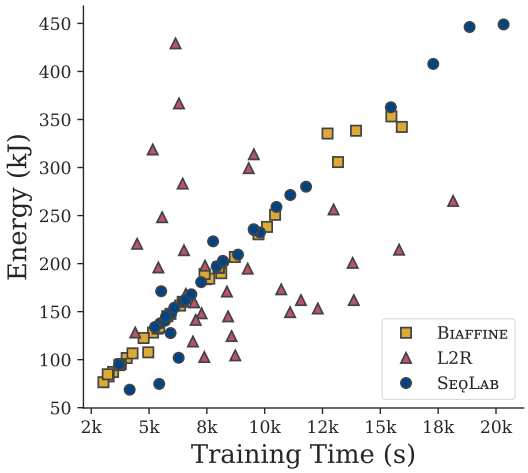

Figure 3: Training energy consumption with respect to training time.

the data, which we expand on in Appendix F.

4 Limitations of analysis

While our analysis is not ground-breaking or particularly expansive in nature, we do think it is useful in practice and acts as mini-review of the current state of affairs in dependency parsing. However, there are a number of limitations in this study. First, we only look at the parameters associated with the BiLSTMs. We feel this is fairly justified, but it is obviously feasible that varying these parameters and not the others could have different effects for each parsing system even if that is fairly unlikely. While we do look at a very diverse set of languages with diverse linguistic features, it is still a fairly small sample. We were somewhat limited by having to train many models and felt it would be better to focus on a sample of diverse languages with quality data than many languages and less model settings. Of course, this analysis could be extended to use more languages, but we expect this would further corroborate the results presented here. Also by using a small set of treebanks, we don't cover a wide array of domains (the data is mainly fiction and news).

Another potential limitation is only using one dependency annotation scheme (the scheme used for CTB was a precursor to UD), but in lieu of a theoretical reason that the parsers would behave differently using a different scheme (e.g. surface syntactic UD (SUD) treebanks containing much more non-projectivity (Gerdes et al., 2018)) this feels like a light limitation.

A slightly more pressing limitation is the absence of a feature analysis because certain systems could potentially benefit from different features. Work has been presented in this direction and has shown that predicted POS tags aren't wonderfully useful (Smith et al., 2018; Anderson and Gómez-Rodríguez, 2020b; Zhang et al., 2020b). However, these analyses didn't include SEQLAB parsers at all and the transition-based system used was a lower-performing system, UUParser. So it is feasible that L2R and SEQLAB would benefit from predicted POS tags. That can be left open for the future.

Another limitation is that we only trained one model for each BiLSTM setting. While training a model for each treebank somewhat offset this, it is still possible that with different initialisation, these parsers would behave slightly differently. However, it is unlikely to cause material differences in the performance and as mentioned, this is quite strongly offset by training on varying treebanks.

And finally, we focused on parsers trained on fairly large amounts of annotated data. We leave the analysis of different parsing systems in a low-resource setting for others, but we point out that when training on very little data, training costs aren't much of a concern and on truly low-resource languages, data parsed at production is also going to be scarce so inference speed won't be the bottleneck.

5 Conclusion

We have presented a simple Pareto optimisation analysis for a representative sample of modern dependency parsers. We evaluated efficiency in two ways. We evaluated the trade-off between accuracy and parsing speed and the trade-off between accuracy and training energy consumption. The BIAFFINE and SEQLAB occupied the speed Pareto front with the former being slower and more accurate and the latter being faster and less accurate. We didn't observe any real trade-off with regards to training energy and performance, but it was clear that SEQLAB is not particularly efficient in this regard. Typically training energy varied based on how long a model took to converge, with L2R being somewhat sensitive to the different treebanks. Overall, for most scenarios, BIAFFINE emerged as a well-balanced practical solution. For the sake of candour, we offer a brief discussion of the limitations of this analysis in Appendix 4.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from ERDF/MICINN-AEI (ANSWER-ASAP, TIN2017-85160-C2-1-R), from Xunta de Galicia (ED431C 2020/11), and from Centro de Investigación de Galicia "CITIC", funded by Xunta de Galicia and the European Union (ERDF - Galicia 2014-2020 Program), by grant ED431G 2019/01.

References

Mark Anderson and Carlos Gómez-Rodríguez. 2020a. Distilling neural networks for greener and faster dependency parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and*

the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies, pages 2–13, Online. Association for Computational Linguistics.

Mark Anderson and Carlos Gómez-Rodríguez. 2020b. On the frailty of universal POS tags for neural UD parsers. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 69–96, Online. Association for Computational Linguistics.

Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, et al. 2017. The hindi/urdu treebank project. In *Handbook of Linguistic Annotation*, pages 659–697. Springer.

Dana Bielec. 1998. *Polish: An Essential Grammar*. Routledge, London.

BIPM. 2019. *Le Système international d'unités / The International System of Units ('The SI Brochure')*, ninth edition. Bureau international des poids et mesures.

Lucien Brown. 2015. Honorifics and politeness. *The handbook of Korean linguistics*, pages 303–319.

Suk-Jin Chang. 1996. *Korean*. John Benjamins, Philadelphia.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Key-Sun Choi, Young S Han, Young G Han, and Oh W Kwon. 1994. Kaist tree bank project for korean: Present and future development. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, pages 7–14. Citeseer.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Jayeol Chun, Na-Rae Han, Jena D Hwang, and Jinho D Choi. 2018. Building universal dependency treebanks in korean. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Federica Cognola and Jan Casalicchio. 2018. On the null-subject phenomenon. *Null subjects in Generative Grammar. A Synchronic and Diachronic Perspective*, pages 1–28.

Bernard Comrie. 1978. Ergativity. In Winfred P. Lehmann, editor, *Syntactic Typology: Studies in the Phenomenology of Language*, pages 329–394. University of Texas Press, Austin.

Mathieu Dehouck, Mark Anderson, and Carlos Gómez-Rodríguez. 2020. Efficient EUD parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 192–205, Online. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. *Proceedings of the 5th International Conference on Learning Representations*.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

Ronald F Feldstein. 2001. *A concise Polish grammar*. Citeseer.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. In *Proceedings of NAACL-HLT*, pages 710–716.

Kim Gerdes, Bruno Guillaume, Sylvain Kahane, and Guy Perrier. 2018. SUD or surface-syntactic Universal Dependencies: An annotation scheme near-isomorphic to UD. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 66–74, Brussels, Belgium. Association for Computational Linguistics.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

One-Soon Her and Chen-Tien Hsieh. 2010. On the semantic distinction between classifiers and measure words in chinese. *Language and linguistics*, 11(3):527–551.

Magnus Jacobsen, Mikkel H. Sørensen, and Leon Derczynski. 2021. Optimal size-performance tradeoffs: Weighing pos tagger models.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

HE Klockmann. 2012. Polish numerals and quantifiers: A syntactic analysis of subject-verb agreement mismatches. Master's thesis, Utrecht University.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas. Association for Computational Linguistics.

Iksop Lee and Robert Ramsey. 2000. *The Korean Language*. State University of New York Press, New York.

Charles N. Li and Sandra A. Thompson. 1981. *Mandarin Chinese: a Functional Reference Grammar*. University of California Press, Berkeley.

Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, and Xiaoyong Du. 2018a. Analogical reasoning on chinese morphological and semantic relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 138–143. Association for Computational Linguistics.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018b. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Meichun Liu. 2015. Tense and aspect in mandarin chinese. *The Oxford Handbook of Chinese Linguistics*, pages 274–289.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414.

Mitchell P Marcus and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

R. S. McGregor. 1977. *Outline of Hindi Grammar*. Oxford University Press, Delhi. 2nd edition.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Martha Palmer, Rajesh Bhatt, Bhuvana Narasimhan, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17.

Waltraud Paul. 2008. The serial verb construction in chinese: A tenacious myth and a gordian knot. *Linguistic Review - LINGUIST REV*, 25:367–411.

G. J. Ramstedt. 1968. *A Korean Grammar*, volume 82 of *Memoirs of the Finno-Ugric Society*. Anthropological Publications, Oosterhout, The Netherlands.

James Robertson. 2006. *The Testament of Gideon Mack*. Penguin Books Ltd.

Anna Siewierska. 1993. Syntactic weight vs. information structure and word order variation in polish. *Journal of Linguistics*, 29:233–265.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018. 82 treebanks, 34 models: Universal Dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123, Brussels, Belgium. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018. An investigation of the interactions between pre-trained word embeddings, character models and POS tags in dependency parsing. In *EMNLP 2018: 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720.

R. Snell and S. Weightman. 1989. *Teach yourself Hindi*. Hodder and Stoughton, London.

Ho-Min Sohn. 1999. *The Korean Language*. Cambridge University Press, Cambridge.

Seok Choong Song. 1988. *Explorations in Korean Syntax and Semantics*. Institute of East Asian Studies, University of California Berkeley, Berkeley.

Drahomíra Johanka Spoustová and Miroslav Spousta. 2010. Dependency parsing as a sequence labeling task. *The Prague Bulletin of Mathematical Linguistics*, 94(2010):7–14.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable dependency parsing as sequence labeling. 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 717–723, Minneapolis, Minnesota. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2020. Bracketing encodings for 2-planar dependency parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2472–2484.

Bernd Wiese. 2011. Optimal specifications: On case marking in polish. *Syntax and morphology multidimensional*, 24:101.

Alina Wróblewska. 2018. Extended and enhanced polish dependency bank in universal dependencies format. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 173–182.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2005. The Penn Chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated chinese corpus. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Xuan Zhang and Kevin Duh. 2020. Reproducible and efficient benchmarks for hyperparameter optimization of neural machine translation systems. *Transactions of the Association for Computational Linguistics*, 8:393–408.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020a. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Yu Zhang, Zhenghua Li, Houquan Zhou, and Min Zhang. 2020b. Is POS tagging necessary or even helpful for neural dependency parsing? *arXiv preprint arXiv:2003.03204*.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408.

Xiyou Zhou, Zhiyu Chen, Xiaoyong Jin, and William Yang Wang. 2021. HULK: An energy efficiency benchmark platform for responsible natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 329–336, Online. Association for Computational Linguistics.

Appendix A Parsers

Left-to-right pointer network (L2R) is a parser which uses a left to right transition-based algorithm that builds a number of attachments equal to the length of a given sentence together with a pointer network which can point to a given position in the sentence for each token (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019).[7] It is one of the current top performing parsers. We use the implementation as is, except we make moderate alterations to overcome hard-coded filepaths and the like. Otherwise, the only hyperparameter we change is the number of encoder layers and the number of nodes in the encoder and decoder layers.

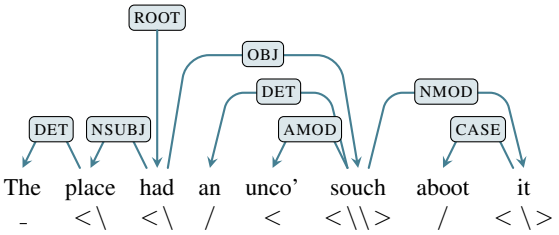

Figure 4: The bracketing encoding from Strzyz et al. (2019). Text is an extract from Robertson (2006).

Sequence labelling parser (SEQLAB) is a parsing system that first encodes trees as a set of labels, so that a direct one-to-one prediction can be made for each token in a sentence (Spoustová and Spousta, 2010; Li et al., 2018b; Strzyz et al., 2019).[8] We use the original bracketing encoding from Strzyz et al. (2019) as it doesn't require UPOS tags to decode (as the other leading encoding does), it performs closely to a more recent bracketing encoding that covers more non-projectivity (Strzyz et al., 2020), and the latter encoding wasn't publicly available when this work commenced. It casts a tree as series of tags which are made up of left and right brackets and forward and backwards slashes which encode the incoming and outgoing arcs for each respective node. The encoding for each token is based on edges associated with the preceding tokens and the direction of the edges. More formally, the encoding for w_i is given by:

$$
\begin{aligned}
&< — \quad \text{if } \epsilon_{j(i-1)} \in \mathcal{E} \wedge j > i-1 \\
&\backslash — \quad \times k \mid k = \sum_{w_j \in S} \begin{cases} 1 & \text{if } j < i \wedge \epsilon_{ij} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \\
&/ — \quad \times k \mid k = \sum_{w_j \in S} \begin{cases} 1 & \text{if } i-1 < j \wedge \epsilon_{(i-1)j} \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \\
&> — \quad \text{if } \epsilon_{ji} \wedge j < i
\end{aligned}
$$

[7] https://github.com/danifg/SyntacticPointer

[8] We use refactored encoding/decoding functions from https://github.com/mstrise/dep2label.

We use the biaffine implementation described below and edit it to be a simple sequence-labelling system, i.e. an embedding layer, followed by a number of BiLSTM layers, and MLPs one for predicting the bracket tags and one for predicting the edge labels. We use the same hyperparameters as used for the biaffine parser.

Deep biaffine (BIAFFINE) is a graph-based parser that creates two representations of each token from the hidden representations from BiLSTMs, hypothesised to be a representation of each token as dependents and as heads (Dozat and Manning, 2017).[9] An affine transformation is applied to the head representation and then this and the dependent one are then combined via a second affine transformation (hence biaffine) to give a matrix of scores, which gives a probability distribution for each node representing the probability any other node is that node's head. A well-formed tree is then enforced using the Chu–Liu/Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967). The edge labels are then predicted based on the predicted edges. We use the standard hyperparameters for this system except where we match them to better correspond to the L2R parser and then only alter the hyperparameters associated with the BiLSTMs.

Appendix B Network hyperparameters

Hyperparameter	Value
Word embedding dimensions	100
Character embedding in (¬ L2R)	32
Character embedding out (¬ L2R)	100
Character dimension (if L2R)	100
Embedding dropout	0.33
Arc MLP dimensions (¬ SEQLAB)	512
Label MLP dimensions (¬ SEQLAB)	128
MLP layers	1
Epochs	200
Patience	10
Training batch size	32

Table 2: Hyperparameters for all models. L2R uses a CNN char. embedding layer and SEQLAB doesn't have a biaffine layer. Other parameters are as in the original (except SEQLAB which uses those of BIAFFINE).

Appendix C Data

We use a small sample of treebanks covering languages from 3 different language families and 4 sub-families and which represent different syntactic systems covering analytic, fusional, and agglutinative languages and all are written in different scripts. We offer a brief description of the treebanks used and some of the salient features of their respective languages. The treebanks were chosen to represent varying syntactic features, but also because of their high quality from being either manually annotated or manually corrected. We also chose relatively large treebanks. The statistics for each treebank are shown in Table 3.

UD Hindi-HDTB (Hindi) is a UD treebank for Hindi based on manually annotated news data (Palmer et al., 2009; Bhat et al., 2017). Hindi is a lightly fusional language with some degree of verbal inflection and noun declension but also makes extensive use of postpositions (McGregor, 1977). It is a split-ergative language meaning in certain cases it uses a nominative-accusative structure but in others it uses an ablative-ergative syntax where the subject of an intransitive verb behaves like the object of a transitive one (Comrie, 1978). It also exhibits tripartite behaviour in certain clauses, where the subject of intransitive verbs, the object of transitive verbs, and the subject of transitive verbs all have different case markings (Comrie, 1978). It is a SOV language, but it has a fairly free word order (Snell and Weightman, 1989). It is Indo-Iranian and is written in the Devanagari script.

UD Polish-PDB (Polish) is a UD treebank manually annotated on fiction, non-fiction, and news data (Wróblewska, 2018). Polish is a highly fusional language with a high degree of verbal inflection (Feldstein, 2001) and 7 case-markings (Wiese, 2011). It is a null-subject language (Cognola and Casalicchio, 2018) with a nominal SVO order but has relatively free word order (Siewierska, 1993). Like most Slavic languages it doesn't make use of articles (Bielec, 1998) but it does have a complex system of numeral and quantifiers that result in agreement mistmatches (Klockmann, 2012). It is a Balto-Slavic language written in the Latin script.

UD Korean Kaist (Korean) is a large treebank generated from a constituency treebank which was semi-automatically annotated with manual corrections based on academic, fiction, and news data (Choi et al., 1994; Chun et al., 2018). Korean is a strongly suffixing agglutinative language (Ramstedt, 1968; Sohn, 1999). This results in a large number of cases and a high degree of verbal inflection (Chang, 1996; Song, 1988; Lee and Ramsey,

[9] The original repository (https://github.com/zysite/biaffine-parser) redirects to a larger set of biaffine based parsers, but is largerly the same.

	Training					Development					Test				
	Sents.	Tokens	Avg. Len.	NP	Chars.	Sents.	Tokens	Avg. Len.	NP	Chars.	Sents.	Tokens	Avg. Len.	NP	Chars.
Chinese	15K	408K	28.2	0.0	4.9	2K	51K	28.0	0.0	4.9	2K	49K	27.3	0.0	4.9
Hindi	13K	281K	22.1	2.6	11.4	2K	35K	22.2	2.4	11.5	2K	35K	22.2	2.4	11.3
Korean	23K	296K	13.9	4.5	8.2	2K	25K	13.2	4.7	8.6	2K	28K	13.4	4.0	8.3
Polish	18K	282K	16.9	1.4	5.4	2K	35K	16.7	1.5	5.4	2K	34K	16.2	1.4	5.4

Table 3: Treebank statistics: number of sentences (Sents.), number of tokens (Tokens), average sentence length (Avg. Len.), percentage for non-projective arcs (NP), average word length (Chars.).

2000). It is technically a SOV ordered languae but it has a highly flexible word order (Ramstedt, 1968; Sohn, 1999). Korean also uses honorifics and speech levels, the former encoding the social relationship between the speaker and the referents in a discussion and the latter the speaker and the person/people being spoken to (Brown, 2015). It is a Koreanic language written in the Hangul script.

Chinese Penn Treebank (Chinese) is large manually annotated treebank for Mandarin based on news data (Xue et al., 2002, 2005). It is an analytic, isolating language with a SVO dominant word order and is a pro-drop language (Li and Thompson, 1981). Chinese has no grammatical tense markers so relies on context or temporal expressions, but aspect is expressed via the use of particles (Liu, 2015). Classifiers and measure words must be used when a noun is preceded by a number, a demonstrative pronoun, or certain quantifiers which are particles that appear between these qualifiers and their respective nouns (Her and Hsieh, 2010). Chinese is said to be a verb stacking language, where more than one verb or verb phrases are stacked together in the same clause, but there is some disagreement if the way verbs are combined actually constitutes verb stacking (Li and Thompson, 1981; Paul, 2008). It is a Sino-Tibetan language written in simplified Hanzi. We re-split the data because the standard split has tiny development and test sets. The resulting sizes are shown Table 3.

Appendix D Training time

Figure 6 shows the average training time (across treebanks) for each parser against the BiLSTM structure. There is a clear linear relation as the complexity of the BiLSTM increases. That is considering a BiLSTM with 2 layers and 1000 nodes to be less complex than one with 3 layers and 400 nodes. We also show a similar plot in the Figure 5, but against the total number of parameters in the network, which shows a similar but less clear trend.

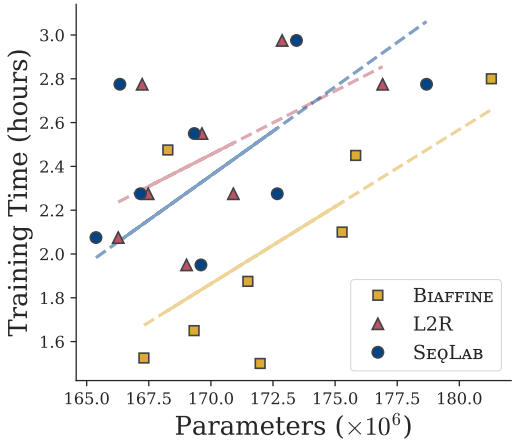

Figure 5: Average training time against total network parameters.

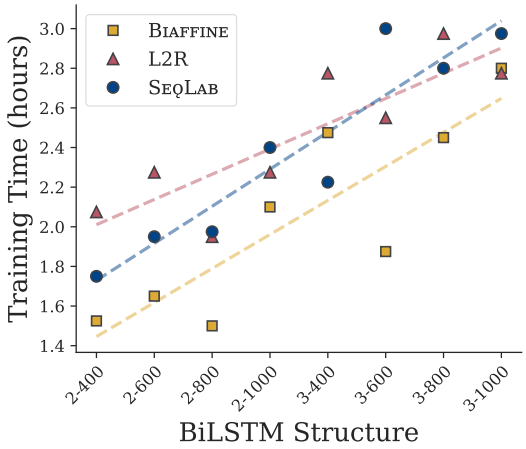

Figure 6: Average training time against BiLSTM structure.

Appendix E Full data

Table 4 shows the full LAS scores for each system for each treebank with different BiLSTM configurations on the development data. Similarly, Table 5 shows the results for the test data. Figure 7 shows LAS against inference speed for the test data and echoes what was observed for the development data in Figure 1. Table 6 shows the total training energy cost, total training time, and the parameters for each parser and for each BiLSTM configuration.

BiLSTM Layers	Nodes	BIAFFINE zh	hi	ko	pl	avg	SEQLAB zh	hi	ko	pl	avg	L2R zh	hi	ko	pl	avg
2	400	80.59	89.83	85.35	86.22	85.50	71.35	85.18	80.47	79.88	79.22	79.68	89.99	83.81	85.32	84.70
	600	81.33	90.48	85.43	86.88	86.03	72.76	86.11	81.05	81.55	80.37	80.62	90.31	84.11	86.54	85.40
	800	81.81	90.61	86.02	87.38	86.45	74.27	86.48	81.63	82.60	81.24	81.59	90.03	83.97	86.77	85.59
	1000	82.15	90.56	85.91	87.95	86.64	74.82	87.06	81.55	82.91	81.58	81.66	90.54	84.06	87.45	85.93
3	400	81.71	90.55	85.45	86.83	86.13	73.62	86.42	81.23	82.15	80.85	82.50	90.85	84.67	87.16	86.29
	600	81.93	90.62	86.04	87.66	86.56	75.46	87.32	81.64	83.03	81.86	83.25	91.13	84.75	87.83	86.74
	800	82.65	91.06	85.94	88.35	87.00	76.20	87.65	81.66	83.90	82.35	83.57	91.00	84.99	88.66	87.06
	1000	82.98	91.16	86.03	88.64	87.20	76.74	87.50	81.61	84.21	82.51	83.41	91.20	85.28	88.84	87.18

Table 4: Full LAS results on the development data.

BiLSTM Layers	Nodes	BIAFFINE zh	hi	ko	pl	avg	SEQLAB zh	hi	ko	pl	avg	L2R zh	hi	ko	pl	avg
2	400	81.03	89.74	84.58	86.76	85.53	72.92	86.66	80.11	82.68	80.59	79.95	90.27	83.13	85.39	84.69
	600	81.82	90.39	84.89	87.38	86.12	74.43	87.59	80.84	83.84	81.68	80.74	90.43	83.77	86.68	85.41
	800	82.27	90.60	85.10	88.20	86.55	75.27	87.74	80.93	84.43	82.09	81.73	90.27	83.57	86.76	85.58
	1000	82.70	90.71	84.83	88.44	86.67	76.44	87.71	80.40	85.07	82.41	81.86	90.35	83.73	87.42	85.84
3	400	81.93	90.42	84.51	87.49	86.09	76.14	88.21	81.40	85.58	82.83	82.63	90.98	84.57	87.59	86.44
	600	82.27	90.23	85.45	88.39	86.59	78.13	88.77	81.96	86.69	83.89	83.69	91.22	84.10	88.09	86.78
	800	83.11	91.08	85.46	88.78	87.11	78.67	88.61	81.75	86.88	83.98	83.72	90.93	84.43	89.18	87.06
	1000	83.47	90.94	85.56	88.86	87.21	78.91	89.26	81.68	87.20	84.26	83.65	91.18	84.47	89.34	87.16

Table 5: Full LAS results on the test data.

BiLSTM Layers	Nodes	Total Energy (MJ) BIAFFINE	SEQLAB	L2R	Total Time (hours) BIAFFINE	SEQLAB	L2R	Avg. Parameters ($\times 10^6$) BIAFFINE	SEQLAB	L2R
2	400	0.54	0.59	0.69	6.1	7.0	8.3	167.3	165.4	166.3
	600	0.58	0.63	0.75	6.6	7.8	9.1	169.3	167.2	167.5
	800	0.53	0.68	0.65	6.0	7.9	7.8	172.0	169.6	169.0
	1000	0.73	0.83	0.76	8.4	9.6	9.1	175.3	172.7	170.9
3	400	0.83	0.81	0.92	9.9	8.9	11.1	168.3	166.3	167.2
	600	0.66	0.96	0.84	7.5	12.0	10.2	171.5	169.3	169.6
	800	0.82	0.98	1.00	9.8	11.2	11.9	175.8	173.4	172.9
	1000	1.01	1.07	0.92	11.2	11.9	11.1	181.3	178.7	176.9

Table 6: Total energy consumed during training, total training time, and average parameters for each parser system for different BiLSTM configurations.

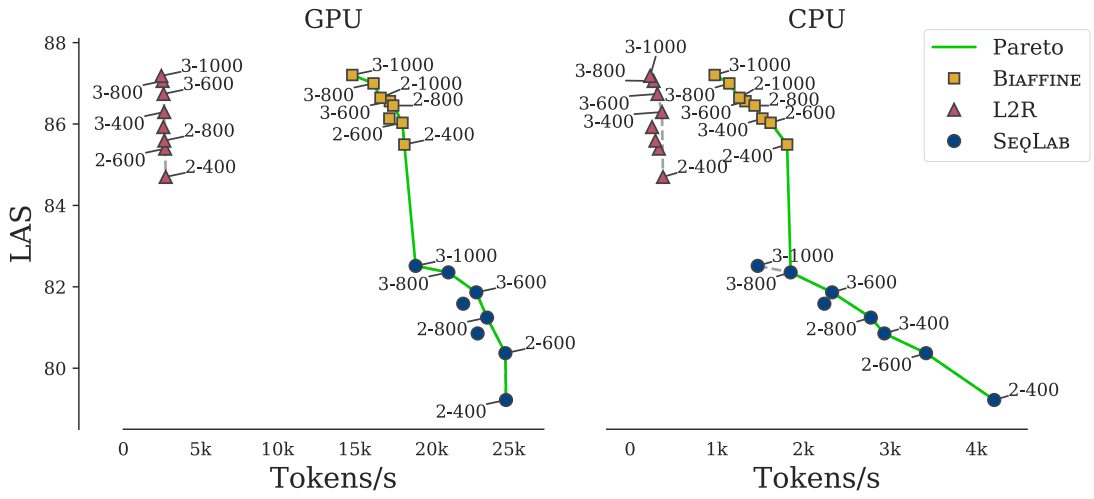

Figure 7: Pareto fronts for L2R, BIAFFINE, and SEQLAB on the test data.

Appendix F L2R training efficiency

Figure 8 shows training energy against training time for L2R for each treebank used. Clearly, the points associated with each treebank cluster. It is clear training the parser on the Korean data is much more energy consuming compared to the others (which form a linear dispersion). It isn't particularly clear why this would be the case based on the statistics in Table 3, except that Korean has the largest number of instances.

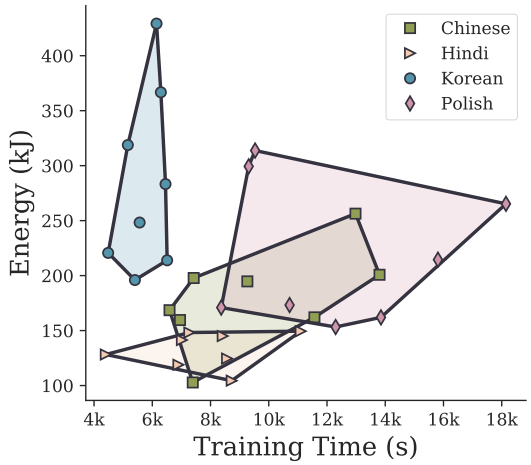

Figure 8: Energy against training time for L2R systems. L2R is more impacted by different data.