Arc Representation for Graph-based Dependency Parsing

Anonymous ACL submission

Abstract

In this paper, we address the explicit representation of arcs in graph-based syntactic dependency parsing, departing from conventional approaches where parsing algorithms compute dependency arc scores directly from input token representations. We propose to augment the parser with an intermediate arc representation, arguing for two main advantages. Firstly, arc vectors encapsulate richer information, improving the capabilities of scoring functions. Secondly, by introducing refinement layers, we allow interactions between arc representations, facilitating interactions between arcs. We demonstrate the efficacy of this approach through evaluations on PTB and UD treebanks. Our approach achieves an LAS error rate reduction of 1.0% on the PTB test set, and 1.7% on UD, over the best SOTA model.

1 Introduction

Recent graph-based dependency model with powerful neural extractors pioneered in (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017) and extended in (Zhang et al., 2020) make the assumption that the plausibility of a lexical arc or its labelling, as expressed by a score, can be computed directly from the vector representation of the two words linked by this arc. This approach has led to tremendous improvements in parsing accuracy, and consequently this assumption has rarely been questioned with the exception of (Ji et al., 2019) where the structure of the parse forest is exploited to rescore arcs, similarly to forest rerankers for statistical parsers (Huang, 2008).

We challenge this assumption through the lens of deep learning. We propose to learn how to *represent* lexical arcs by vectors, and derive scores from these vectors. This method allows to manipulate these vectors through deep architectures without building the parse forest, and we test this hypothesis with transformers. Moreover, while the previous approach is implemented as two pipelines, one for arc scoring and one for arc labelling, sharing word embeddings only, our approach is built around a unique pipeline from word embedding to arc embedding: only the last steps, arc scoring and arc labelling, are specialized. This enforces the sharing of parameters between the two tasks.

2 Model

We review the standard biaffine parser and then highlight the key differences of our arc-centric approach. Prior to parsing, in all systems, from an input sentence $x_0 x_1 \ldots x_n$, where x_0 is the dummy root and $\forall 1 \leq i \leq n$, x_i corresponds to the i^{th} token of the sentence, models start by computing contextual embeddings $e_0, e_1, \ldots, e_n$. This can be implemented in various ways, *e.g.* with pretrained static word embeddings followed by LSTMs, or averaged layers from pretrained dynamic word embeddings. These contextual embeddings are further specialized for head and modifier roles. This is implemented as two feed-forward transformations, resulting in two sets of word representations, $h_0, h_1, \ldots, h_n$ for heads and $m_0, m_1, \ldots, m_n$ for modifiers. In the remainder, given a vector v of size d we note v' the vector of size $d+1$ where $v[i] = v'[i], \forall 1 \leq i \leq d$ and $v'[d+1]$ is set to 1.

2.1 Biaffine Model

We present the local and first-order models as introduced in (Dozat and Manning, 2017) and refer readers to (Zhang et al., 2020) for higher-order extensions. The first-order scoring function decomposes the score of a parse tree as the sum of the scores of its arcs, if they form a valid tree (*i.e.* rooted in x_0, connected and acyclic) and can be implemented as a CRF where arc variables are independently scored but connected to a global factor asserting well-formedness constraints. This CRF can be trained efficiently and inference is performed with well-known algorithms. Still, learning imposes to compute for each sentence *its partition*, the sum of the (exponentiated) scores of all parse candidates. While being tractable, this is an overhead compared to computing arc scores independently without tree-shape constraints. Hence, several recent parsers, *e.g.* (Dozat and Manning, 2017) which called this model *local*, simplify learning by casting it as a head-selection task for each word, *i.e.* arc score predictors are trained without tree constraints. In all cases, tree-constrained CRF or head selection, evaluation is performed by computing the highest-scoring parse (Eisner, 1997; Tarjan, 1977), where arc scores may be replaced by marginal log-probabilities (Goel and Byrne, 2000).

Arc Scores are computed by a biaffine[1] function: for arc $x_i \to x_j$, Dozat and Manning (2017) define arc score as $s_{ij} = h_i^\top M m'_j$ with trainable matrix M. For embeddings of size d, M has dimensions $d \times (d+1)$.[2]

Arc Labelling is considered a distinct task: at training time arc labelling has its own loss and at prediction time most systems use a pipeline approach where first a tree is predicted, and second each predicted arc is labelled.[3] Scoring is also implemented with a biaffine model: for arc $x_i \to x_j$, the label logit vector is $l_{ij} = {h'_i}^\top L m'_j$, with trainable L. For word vectors of size d and for a system with k possible arc labels, L has dimension $(d+1) \times k \times (d+1)$. While we used h and m notations, these specialized word embeddings are given by feed-forward networks different from the ones used for arc scores.

This model relies on two biaffine functions, the first for arc scores returning a scalar per arc, and a second for arc labels scores returning for each arc a vector of label scores. Parameter sharing between arc score and arc labelling computations is limited to contextual word embeddings e.

2.2 Arc Models

Our models differ architecturally in two ways: *(i)* an intermediate vector representation is computed for each arc and *(ii)* both arc and labelling scores are derived from this single arc representation.

For arc $x_i \to x_j$ we compute vector representation v_{ij}. Again, we use a biaffine function outputting a vector similarly to arc labelling in standard models: $v_{ij} = h_i^\top R m'_j$ for a trainable tensor R with dimensions $d \times r \times d$, where r is the size of the arc vector representation v_{ij}, and is a hyperparameter to be fixed, as is the word embedding size. We recover arc score s_{ij} and arc labelling l_{ij} from v_{ij} by feed-forward transformations: $s_{ij} = F_s(v_{ij})$ and $l_{ij} = F_l(v_{ij})$. Note that there is only one biaffine transformation, and one specialization for head and modifier roles. Finally, remark that this change does not impact the learning objective: parsers are trained the same way.

2.3 Refining with Attention

Arc vectors obtained as above can read information from sentence tokens via contextual embeddings. But we can go further and use Transformers (Vaswani et al., 2017) to leverage attention in order to make arc representations aware of other arc candidates in the parse forest and adjust accordingly, effectively refining representations and realizing a sort of forest reranking. We call v_{ij}^0 the vector computed by the biaffine function over word embeddings described above. Then we successively feed vectors of the form v_{ij}^{p-1} to Transformer encoder layer T^p in order to obtain v_{ij}^p and eventually get the final representation v_{ij}^P. This representation is the one used to compute scores with F_s and F_l. Remark again that this change in the vector representation is compatible with any previously used learning objectives.

The main issue with this model is the space complexity. The softmax operation in Transformers requires multiplying all query/key pairs, the result being stored as a $t \times t$ matrix, where t is the number of elements to consider. In our context, the number of arc candidates is quadratic in the number of tokens in the sentence, so we conclude that memory complexity is $O(n^4)$ where n is the number of tokens. To tackle this issue, we could take advantage of efficient architectures proposed recently *e.g.* Linear Transformers (Qin et al., 2022). Preliminary experiments showed training to be unstable so we resort to a simpler mechanism.

Filtered Attention One way to tackle the softmax memory consumption is to filter input elements. If the number of queries and keys fed to the

[1] We ignore bias terms for simplicity.

[2] This additional 1 on the modifier side intuitively makes the expression for s_{ij} mimic the conditional probability of the presence of arc $i \to j$ *given* i is classified as a head word, see (Dozat and Manning, 2017) for a detailed discussion.

[3] We remark that Zhang et al. (2021) learn the two separately and merge them at prediction time.

transformer is linear, we recover a quadratic space complexity. To this end we implement a simple filter F_f to retrieve the best k head candidates per word, reminiscent of some higher-order models prior to deep learning, *e.g.* Koo and Collins (2010) which used arc marginal probabilities to perform filtering. We keep the k highest-scoring $F_f(v^0_{ij})$ for each position j, where k typically equals 10. Kept vectors v^0_{ij} are passed through the transformer as described above, while unkept vectors are considered final. This means that the transformer only processes arcs whose filter scores are among the highest-scoring ones, the intuition being that transformers are only used on ambiguous cases where more context is required to further refine arc or label scores. Details on the filter's implementation can be found in Appendix D.

3 Experiments

Data We conduct experiments on the English Penn Treebank (PTB) with Stanford dependencies (de Marneffe and Manning, 2008), as well as Universal Dependencies 2.2 Treebanks (UD; Nivre et al. 2018), from which we select 12 languages, pseudo-projectivized following (Nivre and Nilsson, 2005). We use the standard split on all datasets. Contextual word embeddings are obtained from RoBERTa$_{\text{large}}$ (Liu et al., 2019) for the PTB and XLM-RoBERTa$_{\text{large}}$ (Conneau et al., 2020) for UD.

Evaluation Metrics We use unlabeled and labeled attachment scores (UAS/LAS), with the latter to select best models on development sets. Results are averaged over 8 runs initialized with random seeds. Following Zhang et al. (2020) and others, we omit punctuations during evaluation on PTB but keep them on UD. Finally, we use gold POS tags on UD but omit them for PTB.

Models LOC is the local model from (Zhang et al., 2020) trained with arc cross-entropy while CRF2O is their second-order CRF. ARCLOC is our model with arc vectors trained with arc cross-entropy. All models are evaluated with the Eisner algorithm (Eisner, 1997) extended to higher-order for CRF2O. We tested 3 parameter regimes: small, standard and large. For LOC, we set standard as the number of parameters used in (Zhang et al., 2020), about 2 million parameters. In the small regime, we halve the size of word vectors and in the large regime we double it. For ARCLOC without transformers, we set word vectors and arc vectors to the same size, that we adjust to reach the same number as for LOC. More details on hyperparameters and the precise definition of the number of parameters are given in Appendix A.

	# Param (10^6)	UAS	LAS
Wang and Tu (2020)⋆		96.94	95.37
Gan et al. (2022) Proj⋆		97.24	95.49
Yang and Tu (2022)⋆⋆		97.4	95.8
Amini et al. (2023) w/o POS ⋆⋆		97.4	95.8
LOC	3.8	97.34	95.88
CRF2O	3.5	97.30	95.86
ARCLOC no transf.	3.3	97.38	95.91
ARCLOC 1 transf. layer	3.5	**97.40**	**95.93**

Table 1: Results on PTB test with RoBERTa, except for ⋆⋆. For last four, average of 8 runs. ⋆: from (Gan et al., 2022). ⋆⋆: from (Amini et al., 2023), using XLNet.

3.1 Main Results

We first evaluate our model on PTB and compare it with other systems trained with RoBERTa. Results in Table 1 show that our approach with arcs represented by their own vector gives a slight performance improvement on both evaluation metrics over LOC a very strong baseline compared to other state-of-the-art parsers. We remark that on PTB higher-order does not help with performance and that the transformer that we hypothesize to encourage arc interactions has a modest beneficial impact.

For the 12 tested languages from UD Table 2 reports results from our parsers and others using XLM-RoBERTa for word embeddings. On average, our model with arc representations achieves state-of-the-art results. On eight languages out of twelve our parser achieves better performance than LOC, or any parser other than the higher-order CRF2O, while keeping a comparable parameter count. We notice that on UD, both explicit higher-order parsing and the use of transformers allow for better results. By increasing the number of parameters in ARCLOC we manage to catch up to CRF2O and achieve state of the art performances in 6 of the 12 languages, at little cost in parsing speed (see Appendix B). Detailed results are given in Appendix E and an error analysis can be found in Appendix F.

3.2 Ablation study

Arc representation As we see in Table 3, arc representations can achieve better performance than the base model, and we notice an increase in the UAS/LAS correlated with an increase in arc

	# Param (10^6)	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg
(Wang and Tu, 2020)		91.42	93.75	92.15	82.20	90.91	92.60	89.51	93.79	91.45	91.95	86.50	92.81	90.75
(Gan et al., 2022) Proj		93.61	94.04	93.10	84.97	91.92	92.32	91.69	94.86	92.51	94.07	88.76	94.66	92.21
(Gan et al., 2022) NProj		93.76	94.38	93.72	**85.23**	91.95	92.62	91.76	94.79	92.97	94.50	88.67	95.00	92.45
LOC	3.8	94.56	94.52	94.14	84.25	92.31	93.88	91.66	94.99	94.11	95.08	90.27	95.81	92.96
CRF2O	3.5	**94.61**	**94.72**	94.17	84.53	92.33	**94.03**	**91.78**	95.06	94.16	95.40	90.22	95.91	93.07
ARCLOC 0 TRANSF. LAYER	3.3	94.28	94.45	94.28	84.17	92.32	93.92	91.65	94.89	94.06	95.11	**90.37**	95.85	92.94
ARCLOC 0 TRANSF. LAYER	50	94.42	94.53	94.32	84.35	92.32	93.96	**91.78**	94.96	94.09	95.12	90.35	95.88	93.00
ARCLOC 1 TRANSF. LAYER	3.5	94.32	94.58	**94.34**	84.43	92.32	93.95	91.64	94.93	94.13	95.45	90.33	95.89	93.03
ARCLOC 2 TRANSF. LAYER	50	94.40	94.62	**94.34**	84.54	**92.39**	94.00	91.75	**95.08**	**94.18**	**95.52**	90.30	**95.93**	**93.09**

Table 2: Test LAS for 12 languages in UD2.2. We use ISO 639-1 codes to represent languages.

size up to a plateau.

Size of Arc Vector	# Param (10^6)	UAS	LAS
N/A (LOC)	2	96.79	95.10
32	8	96.85	95.20
64	16	96.89	95.24
128	32	96.90	**95.26**
256	64	96.89	95.24
512	128	**96.92**	95.25

Table 3: PTB dev scores w.r.t. arc vector sizes, word vector size set to 500[4](8 run average, no transformer).

Role of Attention We run an ablation experiment to measure the impact of the Transformer module in our architecture. Table 4 shows that our arc representation is the main factor of improvement of the baseline LOC for PTB.

	# Param (10^6)	UAS	LAS
LOC	3.8	96.83	95.11
CRF2O	3.5	96.85	95.15
ARCLOC 0 Transf. layer	3.3	96.86	95.21
ARCLOC 1 Transf. layer	3.5	96.89	95.24

Table 4: Impact of arc vectors and Transformers on PTB dev data.

4 Related Work

In addition to the encoder, attention is widely utilized in syntactic analysis (Mrini et al., 2020; Tian et al., 2020). For instance, Kitaev and Klein (2018) examine the correlation between attention on lexical and positional contents, while Le Roux et al. (2019) employ specialized cross-attention for transition-based parsing. Representing spans has been shown to be beneficial for NLP (Li et al., 2021; Yan et al., 2023; Yang and Tu, 2022) as well as using transformers to enhance them (Zaratiana et al., 2022). Our method is closely related to the use of global attention in Edge Transformers (Bergen et al., 2021). Besides the difference in formalisms of analysis, we do not use any particular attention mask while they use a *triangular* attention. Our use of a filter to select arcs which are allowed to interact, is more flexible. Other novel forms of graph attention have been proposed, *e.g.* in NodeFormer (Wu et al., 2022). Our use of transformers over arcs is a part of a growing literature on generalizing transformers to relational graph-structured data, as called for by Battaglia et al. (2018). This includes approaches that encode graphs as sets and input them to standard transformers similar to TokenGT (Kim et al., 2022) and Graphormer (Ying et al., 2021). However, we restrict the transformer input to arc vectors excluding node. We differ from approaches that modify standard self-attention either to model structural dependencies (Kim et al., 2017) or implement relative positional encodings (Cai and Lam, 2019; Hellendoorn et al., 2020), which do not maintain arc vectors but instead use them to improve node vectors. Lastly, we note that our model bears resemblances to earlier work on reranking for parsing (Collins and Koo, 2005; Le and Zuidema, 2014), as we use transformers to promote or demote arcs before scoring and parsing.

5 Conclusion

We presented a change in the main graph-based dependency parsing architecture where lexical arcs have their own representation in a high-dimensional vector space, from which their lexical scores are computed. This model demonstrates a clear improvement on parsing metrics over a strong baseline and achieves state-of-the-art performance on PTB and 12 UD corpora. Moreover we show that this architecture is amenable to further processing where arc vectors are refined through transformers and allowing interaction between arcs similar to higher-order features. This method could be extended to other tasks, such as constituent parsing or relation extraction.

[4]ArcLoc's param # is higher due to the word vector size of 500 which we generally do not use elsewhere (see A).

6 Limitations

Our system with Transformers relies on the attention mechanism which is quadratic in space and time in the number of elements to consider. Since the number of elements (arcs in our context) is itself quadratic in the number of word tokens, this means that naively the proposed transformer extension is of quadratic complexity. In practice we showed that adding a filtering mechanism is sufficient to revert complexity back to $O(n^2)$, but we leave using efficient transformers, with linear attention mechanism, to future work.

7 Ethical Considerations

We do not believe the work presented here further amplifies biases already present in the datasets. Therefore, we foresee no ethical concerns in this work.

References

Afra Amini, Tianyu Liu, and Ryan Cotterell. 2023. Hexatagging: Projective dependency parsing as tagging. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1453–1464, Toronto, Canada. Association for Computational Linguistics.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Leon Bergen, Timothy J. O'Donnell, and Dzmitry Bahdanau. 2021. Systematic generalization with edge transformers. *CoRR*, abs/2112.00578.

Deng Cai and Wai Lam. 2019. Graph transformer for graph-to-sequence learning.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK. Coling 2008 Organizing Committee.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jason Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 54–65, Boston/Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Leilei Gan, Yuxian Meng, Kun Kuang, Xiaofei Sun, Chun Fan, Fei Wu, and Jiwei Li. 2022. Dependency parsing as MRC-based span-span prediction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2427–2437, Dublin, Ireland. Association for Computational Linguistics.

Vaibhava Goel and William J. Byrne. 2000. Minimum bayes-risk automatic speech recognition. *Comput. Speech Lang.*, 14(2):115–135.

Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. 2020. Global relational models of source code. In *International Conference on Learning Representations*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594. Association for Computational Linguistics.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018, pages 876–885. Association For Uncertainty in Artificial Intelligence (AUAI). Funding Information: Acknowledgements. This work was supported by NSF IIS-1563887, Samsung Research, Samsung Electronics and Russian Science Foundation grant 17-11-01027. We also thank Vadim Bereznyuk for helpful comments. Funding Information: This work was supported by NSF IIS-1563887, Samsung Research, Samsung Electronics and Russian Science Foundation grant 17-11-01027. We also thank Vadim Bereznyuk for helpful comments. Publisher Copyright: © 34th Conference on Uncertainty in Artificial Intelligence 2018. All rights reserved.; 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018 ; Conference date: 06-08-2018 Through 10-08-2018.

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.

Jinwoo Kim, Dat Tien Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. 2022. Pure transformers are powerful graph learners. In *Advances in Neural Information Processing Systems*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *International Conference on Learning Representations*.

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.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739, Doha, Qatar. Association for Computational Linguistics.

Joseph Le Roux, Antoine Rozenknop, and Mathieu Lacroix. 2019. Representation learning and dynamic programming for arc-hybrid parsing. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 238–248, Hong Kong, China. Association for Computational Linguistics.

Fei Li, ZhiChao Lin, Meishan Zhang, and Donghong Ji. 2021. A span-based model for joint overlapped and discontinuous named entity recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4814–4828, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Sandra Bellato, Kepa Bengoetxea, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Shoval Sadde, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Veronika Vincze, Lars Wallin, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.2. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. 2022. The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7025–7041, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

R. E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7(1):25–35.

Yuanhe Tian, Yan Song, Fei Xia, and Tong Zhang. 2020. Improving constituency parsing with span attention. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1691–1703, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. 2022. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, volume 35, pages 27387–27401. Curran Associates, Inc.

Zhaohui Yan, Songlin Yang, Wei Liu, and Kewei Tu. 2023. Joint entity and relation extraction with span pruning and hypergraph neural networks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7512–7526, Singapore. Association for Computational Linguistics.

Songlin Yang and Kewei Tu. 2022. Headed-span-based projective dependency parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2188–2200, Dublin, Ireland. Association for Computational Linguistics.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*.

Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. 2022. GNNer: Reducing overlapping in span-based NER using graph neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 97–103, Dublin, Ireland. Association for Computational Linguistics.

Xudong Zhang, Joseph Le Roux, and Thierry Charnois. 2021. Strength in numbers: Averaging and clustering effects in mixture of experts for graph-based dependency parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 106–118, Online. Association for Computational Linguistics.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. 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.

A Hyperparameters

We mostly use the same hyperparameter settings as Zhang et al. (2020) which are found in their released code. Specifically we adopt the approach they use when training models using BERT, using the average of the 4 last layers to compute our word embeddings. The batch size is 5000, the dimension of the arc MLP is 96, 120, 144 for ArcLoc with 1×10^6, 2×10^6 and 4×10^6 parameters respectively and for Loc it's 256, 500, 900 and the label MLP dimension is 64, 100, 140 for the Loc 1×10^6, 2×10^6, and 3 or 4×10^6 models, also respectively, for the model with 50×10^6 parameters we use an arc MLP dimension of 500 and an arc size of 192. Our transformer uses a number of attention heads always equal to one sixteenth of the arc size, except when we have an arc size of 120 as it is not a multiple of 16, there we use 8 attention heads. The dropout rate for the MLPs with Loc is 0.33 and for ArcLoc it's 0.1 except for the 50×10^6 parameters model where we use a dropout of 0.33 for the arc MLPs and 0.1 for all other MLPs, we train our model for 10 epochs and save the one with the best LAS score on the dev data. The learning rates are 8.3e-6 and 3.7e-5 for Loc and ArcLoc respectively before the stochastic weight averaging (SWA) and 5e-6 and 3.7e-6 also respectively from the fifth epoch onward when we use SWA. The transformer in ArcLoc benefits from its own hyperparameters, while the model warms up for one epoch, the transformer does so for three and has a base learning rate of 3e-3, which becomes 6e-5 when using SWA. For CRF2o we use the exact same hyperparameters as Zhang et al. (2020) except for the learning rates which are the same as Loc. We use the following formula to determine the parameter count for Loc with 2 arc MLPs, 2 label MLPs, and 2 biaffine modules, one for the arcs and one for the labels:

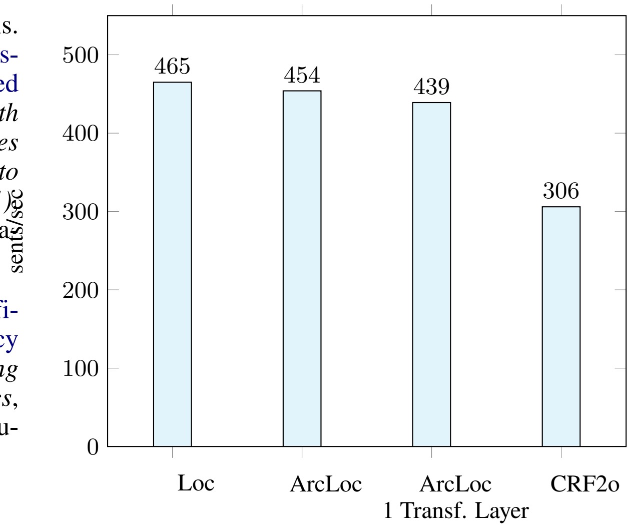

Figure 1: Parsing speed on PTB in sent/s.

$$2 * 1024X + 2 * 1024Y + X^2 + Y^2 L \quad (1)$$

Where 1024 is the size of RoBERTa's and XLM-RoBERTa's output, X and Y are the arc and label MLP dimensions respectively and L is the number of labels in the dataset. For ArcLoc which uses no label MLPs or biaffine but 2 scoring MLPs with hidden sizes of d/2 for the arc scoring and 2L for the label scoring and an arc size of d we have the following:

$$2 * 1024X + X^2 d + \frac{d^2 + d}{2} + 2dL + 2L^2 \quad (2)$$

Additionally, with N as the number of transformer layers used, the transformer adds a total parameter count of:

$$N * 10d^2 \quad (3)$$

B Efficiency

We trained the bulk of our models on Nvidia a100 GPUs with 80GB of memory or v100 GPUs with 32GB of memory. Our models' memory footprint and speed directly depend on the size of the arcs, whether we use a transformer, and whether we filter the arcs. As we can see in figure 1, our model achieves speeds comparable to Loc, and is still faster than CRF2o even with 50×10^6 parameters with a speed of 413 sents/sec, furthermore with the use of our filter, we reduce the memory consumption to manageable levels allowing us to use a softmax transformer.

C Stochastic weight averaging

We implement stochastic weight averaging (SWA) introduced in Izmailov et al. (2018) after 4 epochs, which we found lead to consistent improvements in all models (LOC, ARCLOC, CRF2O) after fine-tuning.

D Filtering Arcs

The filtering step keeps k arcs per modifier. It is inspired from the straight-through estimator (Bengio et al., 2013) and is implemented as follows.

For each token m we compute the scores of all arcs $h \rightarrow m$, from their vector representations v_{hm}. Then we add some Gumbel noise (at training time) and normalize scores via softmax: we obtain probabilities $p(h \rightarrow m)$ that we use to sort arcs from most to least probable: $h_1 \rightarrow m \dots h_n \rightarrow m$.

Finally the k^{th} arc vector returned by the filter for modifier m is computed as:

$$v_k(m) = \text{argsort}(v_{h_1 m} \dots v_{h_n m})[k] - \text{detach}(\mathbb{E}[v_{hm}]) + \mathbb{E}[v_{hm}] \quad (4)$$

During the forward pass the two last terms cancel each other out and $v_k(m)$ is the vector of the k^{th} most probable arc for m, $h_k \rightarrow m$. During the backward pass, the first two terms have zero gradient, and the third one amounts to a weighted average of the vectors of arcs $h_1 \rightarrow m \dots h_n \rightarrow m$, with weights given by their probabilities.

Table 5 compares parsing UAS and the filter's oracle UAS (percentage of correct heads in the set returned by the filter). We keep 10 potential heads per word to get the highest oracle score with a reasonably small sequence of arcs.[5]

#Heads	1	2	3	5	10
Oracle	37.65	75.88	92.48	99.10	99.88
Parser	48.79	78.06	89.69	94.74	96.88

Table 5: PTB Dev UAS scores for ARCLOC and its filter's Oracle with different filter sizes (number of kept heads per word).

E Detailed Results

Table 6 gives a detailed account of the results our different on PTB development set. We see that our model ARCLOC with the same number of parameters as LOC gives an absolute 0.1%. We stress that in the case of PTB our approach leads to a better improvement than what the second-order scoring function can bring. We can also remark that one layer of transformers can in some case (large setting) bring a minor improvement.

Model	# Param (10^6)	Dev	
		UAS	LAS
LOC	0.9	96.80	95.10
LOC	1.9	96.79	95.10
LOC	3.8	**96.83**	**95.11**
CRF2O	3.5	**96.85**	**95.15**
ARCLOC 0 TRANSF. LAYERS	1.1	96.83	95.17
ARCLOC 0 TRANSF. LAYERS	2	96.86	95.20
ARCLOC 0 TRANSF. LAYERS	3.3	**96.86**	**95.21**
ARCLOC 1 TRANSF. LAYER	1.2	96.86	95.20
ARCLOC 1 TRANSF. LAYER	2.1	96.87	95.21
ARCLOC 1 TRANSF. LAYER	3.5	**96.89**	**95.24**
ARCLOC 2 TRANSF. LAYERS	1.3	96.85	95.18
ARCLOC 2 TRANSF. LAYERS	2.3	**96.87**	**95.21**
ARCLOC 2 TRANSF. LAYERS	3.7	96.84	95.19
ARCLOC 4 TRANSF. LAYERS	1.5	96.83	95.17
ARCLOC 4 TRANSF. LAYERS	2.6	**96.87**	95.19
ARCLOC 4 TRANSF. LAYERS	4.1	96.86	**95.21**

Table 6: Dev scores for the PTB for different numbers of parameters per model (in millions) and different numbers of layers for ARCLOC

For UD, we report development results in Table 7. Here the picture is a bit different since we can see that second-order score improves over LOC. Please also notice that on UD, gold POS tags are provided. In this case, we see that ARCLOC struggles to improve over LOC. We conclude that the arc representation on its own cannot replace second-order features, and that our experimental setup may be too restrictive for ARCLOC: to get numbers of parameters comparable with LOC we use smaller word embeddings.

However, we remark that adding transformer layers allows our approach to recover the performance offered by CRF2O. Finally we test a setup where use the same word embedding size that CRF2O one transformer layer of size 192. In this case, we see that our model performs on a par with CRF2O.

F Error Analysis

We restrict our error analysis to English but apply it to both UD and PTB. We conclude from our analysis that the ArcLoc systems improve over the Loc baseline across the majority of error categories we have examined.

[5] Note that there is no discrepancy in the first or second column, we can have a UAS score higher than filter's oracle, as an arc can be filtered out and still end up in the parse, our filter only chooses arcs to be processed by the transformer.

	# Param (10^6)	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg
projective%		99.8	99.6	99.2	97.7	99.6	99.6	99.7	99.8	99.4	99.3	99.4	99.2	99.4
CRF2o	3.5	93.20	94.40	94.53	**89.29**	92.90	94.23	92.93	94.55	**95.37**	95.74	**90.39**	95.34	93.57
Loc	0.9	92.73	94.13	94.41	88.56	92.81	93.97	92.87	94.28	95.12	95.61	90.18	95.22	93.32
Loc	1.9	92.89	94.17	94.45	88.63	92.92	93.99	92.90	94.40	95.20	95.67	90.15	95.22	93.38
Loc	3.8	92.83	94.25	94.48	88.75	92.91	94.07	92.95	94.35	95.19	95.72	90.20	95.24	93.41
ArcLoc 0 Transf. layers	1.1	92.71	94.16	94.51	88.17	92.79	93.91	92.92	94.28	94.92	95.63	90.18	95.28	93.29
ArcLoc 0 Transf. layers	2	92.75	94.24	94.57	88.38	92.89	93.99	92.90	94.29	95.01	95.73	90.19	95.30	93.35
ArcLoc 0 Transf. layers	3.3	92.90	94.24	94.61	88.59	92.94	94.04	92.96	94.39	95.03	95.74	90.19	95.33	93.41
ArcLoc 0 Transf. layers	50	93.01	94.29	94.64	88.70	93.02	94.11	93.00	94.45	95.13	95.73	90.28	95.39	93.48
ArcLoc 1 Transf. layer	1.2	92.79	94.25	94.60	88.40	92.86	93.97	92.90	94.34	94.99	95.79	90.05	95.31	93.35
ArcLoc 1 Transf. layer	2.1	92.89	94.31	94.63	88.55	93.00	94.05	92.98	94.39	95.04	95.81	90.21	95.34	93.43
ArcLoc 1 Transf. layer	3.5	93.06	94.34	94.60	88.77	93.02	94.14	92.97	94.48	95.00	95.86	90.28	95.37	93.49
ArcLoc 1 Transf. layer	50	93.13	94.41	**94.69**	88.82	93.09	94.16	**93.05**	94.58	95.17	**95.91**	90.26	**95.43**	93.56
ArcLoc 2 Transf. layers	1.3	92.89	94.24	94.62	88.42	92.87	94.01	92.93	94.37	95.06	95.78	90.13	95.30	93.39
ArcLoc 2 Transf. layers	2.3	92.98	94.31	94.65	88.61	93.01	94.10	92.98	94.40	95.14	95.81	90.23	95.37	93.46
ArcLoc 2 Transf. layers	3.7	93.03	94.37	94.67	88.85	93.06	94.15	93.00	94.52	95.13	95.86	90.16	95.38	93.52
ArcLoc 2 Transf. layers	50	**93.22**	**94.42**	94.68	89.01	**93.15**	**94.25**	93.04	**94.59**	95.14	95.87	90.26	95.41	**93.59**
ArcLoc 4 Transf. layers	1.5	92.94	94.32	94.65	88.60	92.97	94.10	92.95	94.45	95.03	95.81	90.12	95.34	93.44
ArcLoc 4 Transf. layers	2.6	93.02	94.33	94.66	88.66	92.99	94.13	92.99	94.48	95.06	95.84	90.18	95.36	93.48
ArcLoc 4 Transf. layers	4.1	93.08	94.36	94.67	88.78	93.05	94.15	92.99	94.49	95.06	95.86	90.19	95.36	93.50
ArcLoc 4 Transf. layers	51	93.16	94.40	**94.69**	89.03	93.12	94.21	93.01	94.50	95.16	**95.91**	90.26	95.41	93.57

Table 7: Dev LAS for 12 languages in UD2.2 for different numbers of parameters per model and different numbers of layers for ArcLoc

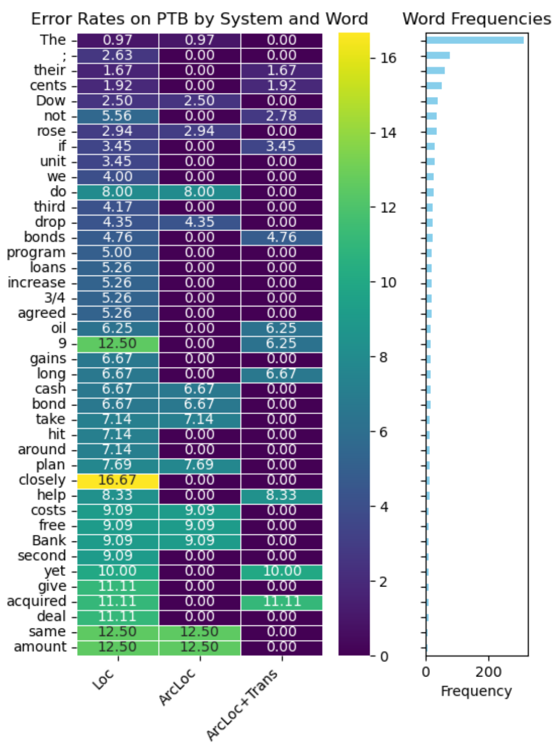

Figure 2: PTB: List of the words with an attachment error rate at least 4 times lower for one of the ArcLoc parsers compared to the Loc parser.

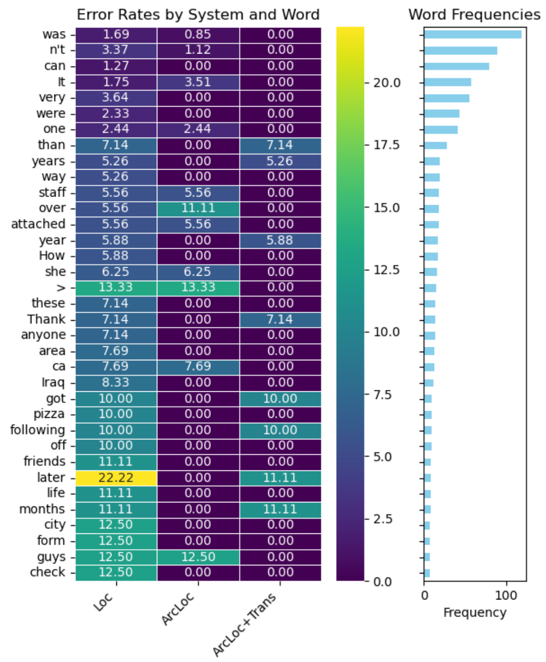

Figure 3: UD: List of the words with an attachment error rate at least 4 times lower for one of the ArcLoc parsers compared to the Loc parser.

F.1 Error Rate per Word Type

In this section we focus on evaluating the performance of various systems across different word frequency groups. Word frequencies were categorized into four groups: '1-5', '6-10', '11-15', and '>15'. Each category represents the range of times words appear in PTB and UD dev corpora.

We employed the Kruskal-Wallis test to assess statistical differences across all systems within each frequency group, followed by pairwise comparisons using the Mann-Whitney U test to evaluate differences between each pair of systems.

Results For UD The Kruskal-Wallis test showed a significant difference between systems in the group ’1-5’ (Statistic=15.858, p<0.001), indicating varying performances among the systems for rare words. Pairwise comparisons revealed significant improvement from Loc and both ArcLoc (p=0.001) and ArcLoc+Trans (p<0.001). However, no significant difference was found between the two ArcLoc parsers (p=0.709), suggesting similar capabilities in handling rare words. No significant differences were observed across systems in the other categories (p=0.070) for the group ’6-10’, (p=0.167) for the group ’11-15’ and (p=0.367) for the group ’>15’. For PTB, we do not observe significant differences for these categories.

To get a sense of the improvements obtained by the ArcLoc parsers, Figures 2 and 3 show the lists of words with an attachment error rate at least 4 times lower for one of the ArcLoc parser compared to the Loc parser.

F.2 Error Rate by Attachment Distance

In this section we evaluate the error rates associated with different attachment distances across the multiple parsing systems. Attachment distance, defined as the number of words between a dependent and its head, helps understand performance on long-range dependencies that are typically hard to model correctly. Figure 4 shows both raw and normalized errors by distance for all parsers on UD dev set.

For small distances, in the range ‘1-3’, all systems perform similarily with statistical difference in error rates. For longer distances in the range ‘>3’, both ArcLoc and ArcLoc+Trans outperform Loc significantly (p=0.01 and p=0.02 respectively). However, we found no significant difference between the two ArcLoc systems.

While similar figures are obtained for PTB and are omitted here, the differences are not statistically significant according to our tests.

F.3 Error Rate by POS

In this section we compare error rates by part-of-speech (POS) tag. Figure 6 displays the result of the comparison across three parsing systems on UD and PTB dev sets using gold POS tags. The error rates are normalized to show the percentage of errors within each POS category.

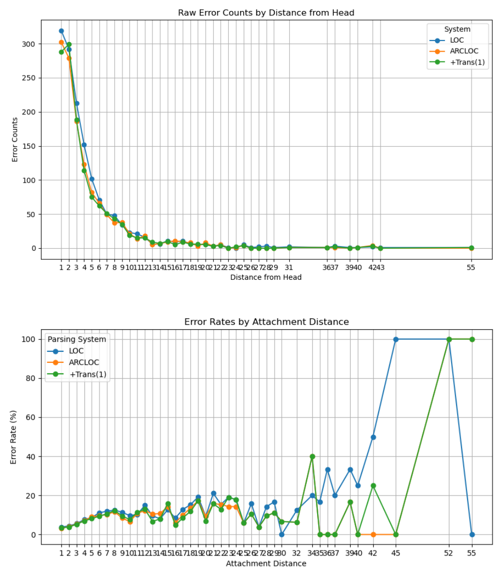

Figure 4: Error counts and rates by attachment distance from a dependent to its head in UD dev test for three systems. All systems are using the Eisner parsing algorithm.

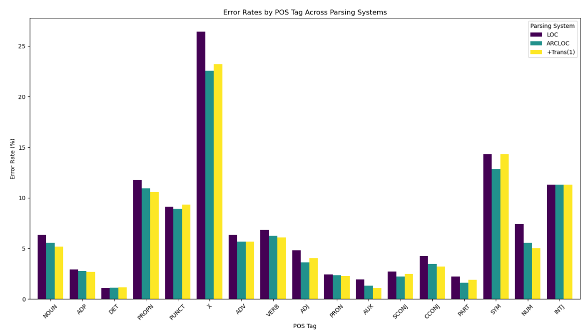

Figure 5: UD: Error rates by gold POS tag for Loc, ArcLoc and ArcLoc+Trans.

For UD, we note that the category ‘X’ (representing a diverse group of tokens) showed the highest error rates across all systems with Loc exhibiting the highest error rate of 26.45%. Conversely, determiners (‘DET’) displayed the lowest error rates. ArcLoc consistently showed lower error rates across most POS tags compared to Loc and +Trans, particularly in ’ADJ’ and ’NUM’ categories. However, while there are variations in error rates across different POS tags, there is no statistically significant difference in the overall error rates between the systems.

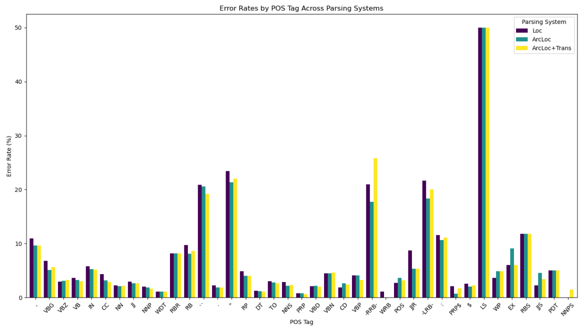

Figure 6: PTB: Error rates by gold POS tag for Loc, ArcLoc and ArcLoc+Trans.

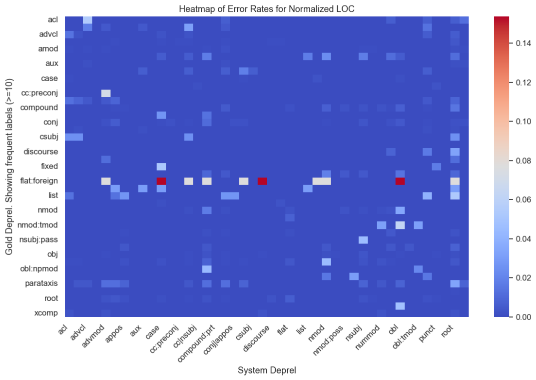

Figure 8: UD: Confusion matrix for Loc.

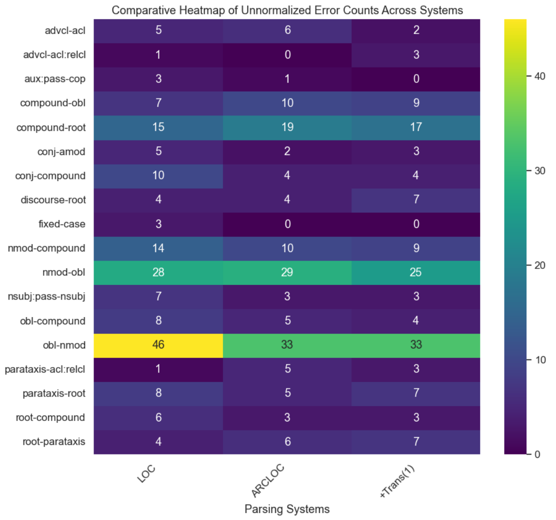

Figure 7: UD: Error counts per dependency type for all systems.

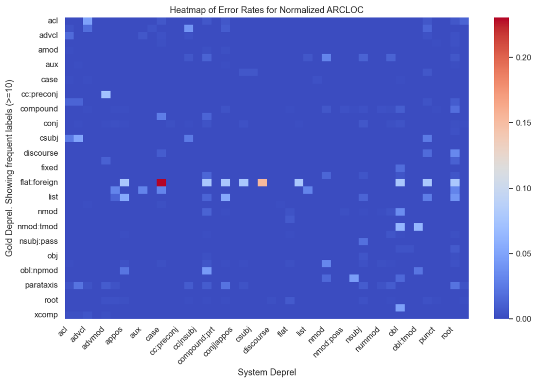

Figure 9: UD: Confusion matrix for ArcLoc.

F.4 Error Analysis by DepRel Type

We examine error distributions among the three parsing systems and find discrepancies in handling specific dependency relations.

Figure 7 show the raw counts for errors for each dependency relation types for all parsers. We note a reduction in errors for the majority of types for the ArcLoc systems compared to the Loc baseline.

Finally, Figures 8, 9 and 10 show the confusion matrices between gold and predicted dependency types for types that appear at least 10 times in the UD dev set. We note that the greatest confusion is related to flat:foreign which amounts to peculiarities in the gold annotation.

F.5 Error Rates by Depth in Gold Tree

Figure 11 illustrates the error rates for three parsers across various depths of dependents in the gold tree. Each depth from 0 to 9 is analyzed, with the frequency of each depth's occurrence provided in parentheses.

We observe that depth 0 has the lowest error rates across all three systems, indicating a higher accuracy in identifying root elements or top-level dependencies. As depth increases to 1 and 2, error rates slightly rise, but the differences between the three systems become more apparent. Depths 5 and 6 also show an increase in error rates especially for the Loc system while both ArcLoc systems suffer less.

The highest error rates are observed at depth 7, especially notable in the Loc system which peaks at 10.55%. However, depth 8 shows a reduction in errors, and surprisingly, depth 9 records a zero error rate for Loc which may be due to the very low frequency of samples at that depth (only 26 instances).

Overall, the ArcLoc systems improve over the Loc baseline across most depths.

Similar patterns can be observed for PTB the Figure 12 but with lower overall error rates which peak at depth 2 for the all systems. In contrast to UD, the error rates on PTB are more uniform across depths and parsing systems, with

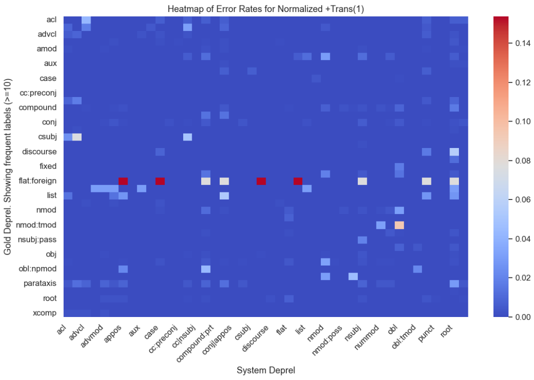

Figure 10: UD: Confusion matrix for ArcLoc+Trans.

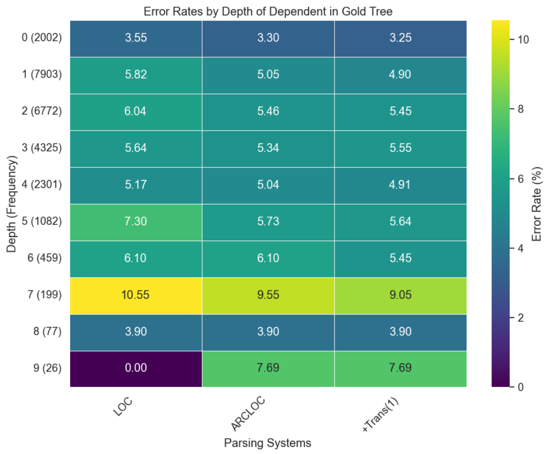

Figure 11: UD: Error Rates by Depth in Gold Tree.

fewer drastic fluctuations. This could indicate a more consistent performance of the parsing systems across various sentence complexities.

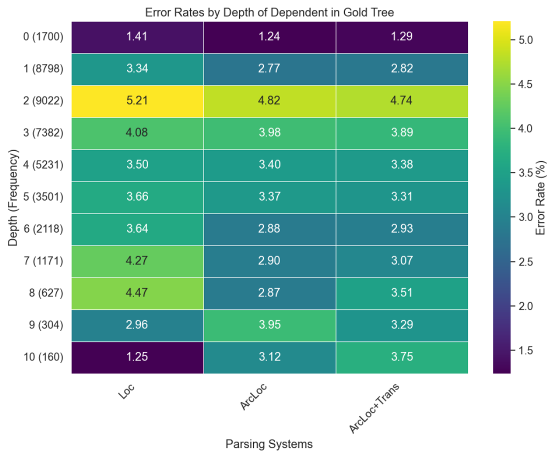

Figure 12: PTB: Error Rates by Depth in Gold Tree.