Dependency Parsing is More Parameter-Efficient with Normalization

Paolo Gajo
University of Bologna
Italy
paolo.gajo2@unibo.it

Domenic Rosati
Dalhousie University
Canada
domenic.rosati@dal.ca

Hassan Sajjad
Dalhousie University
Canada
hsajjad@dal.ca

Alberto Barrón Cedeño
University of Bologna
Italy
a.barron@unibo.it

Abstract

Dependency parsing is the task of inferring natural language structure, often approached by modeling word interactions via attention through biaffine scoring. This mechanism works like self-attention in Transformers, where scores are calculated for every pair of words in a sentence. However, unlike Transformer attention, biaffine scoring does not use normalization prior to taking the softmax of the scores. In this paper, we provide theoretical evidence and empirical results revealing that a lack of normalization necessarily results in overparameterized parser models, where the extra parameters compensate for the sharp softmax outputs produced by high variance inputs to the biaffine scoring function. We argue that biaffine scoring can be made substantially more efficient by performing score normalization. We conduct experiments on six datasets for semantic and syntactic dependency parsing using a one-hop parser and a multi-hop GNN parser. We train N-layer stacked BiLSTMs and evaluate the parser's performance with and without normalizing biaffine scores. Normalizing allows us to achieve state-of-the-art performance with fewer samples and trainable parameters. Code: https://anonymous.4open.science/r/EfficientSDP-7A93

CCS Concepts

• **Computing methodologies** → **Natural language processing**.

Keywords

semantic dependency parsing, biaffine attention, normalization, graph neural networks

ACM Reference Format:
Paolo Gajo, Domenic Rosati, Hassan Sajjad, and Alberto Barrón Cedeño. 2025. Dependency Parsing is More Parameter-Efficient with Normalization. In *Proceedings of the Workshop on Machine Learning on Graphs in the Era of Generative AI (MLoG-GenAI).* ACM, New York, NY, USA, 17 pages.

MLoG-GenAI, Toronto, Canada

1 Introduction

Dependency parsing (DP) consists in classifying node labels t_i (words), edges e_{ij} (relations), and edge labels r_{ij} (relation types) of a dependency graph [30]. A popular model for this task, introduced by Dozat and Manning [8], entails modeling word interactions as a fully connected graph via biaffine attention. Despite its simplicity, a number of models using this biaffine transformation [3, 8, 9, 14, 16] require more parameters than necessary, due to what we identify as a lack of normalization of its outputs. We propose that this overparameterization is caused by the high variance of the outputs, which extra parameters help mitigate. After showing that variance can be reduced through normalization, we demonstrate that similar or better performance can be obtained for DP with fewer parameters and training samples.

In this task, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ comprising a set of nodes $\mathcal{V}$, connected by a set of edges $\mathcal{E}$, with each edge $e_{ij} \in \mathcal{E}$ connecting pairs of nodes $(v_i, v_j) \in \mathcal{V}$. In latent graph inference for dependency graphs, a sentence of $|\mathcal{V}|$ words is modeled as a $|\mathcal{V}| \times |\mathcal{V}|$ graph via biaffine scoring [8]:

$$XWX^\top = X(W_Q W_K^\top)X^\top = XW_Q(XW_K)^\top = QK^\top, \quad X \in \mathbb{R}^{|\mathcal{V}| \times d}.$$

Observe that this is equivalent to the unnormalized scores used in [36]'s self-attention: $QK^\top/\sqrt{d_k}$, where d_k is the output size of the keys and $\text{Attention}(Q, K, V) = \text{Softmax}\left(QK^\top/\sqrt{d_k}\right)V$.

The main difference resides in the fact that Transformer attention scores are scaled by $a = 1/\sqrt{d_k}$. As explained in [36], this scaling is done because high variance inputs to the softmax function will result in large values dominating the output ($e^{x \gg 1}$) and small values decaying ($e^{x \ll 1}$). The downstream effect is exploding and vanishing gradients. To see how this works, observe that the score s between any query and key vector is the result of their dot product $q_i k_i^\top$. Now *if* we assume that these vectors are zero mean and unit variance, then the variance is $\text{Var}(s) = d_k$. Finally, we get our normalization factor, which ensures that each entry in the score matrix has a standard deviation of 1, since $\text{Std} = \sqrt{d_k}$. Consequently, lower input variance will result in more stable outputs.

We observe that there is no consistency in the literature on the use of this scaling term for DP. Most works do not use this scaling term [3, 8, 9, 9, 14, 16], with the exception of [35] who use [36]'s self-attention out of the box. In this paper, we carry out experiments on semantic and syntactic DP tasks, using a wide range of architecture

ablations. We extend the work of Bhatt et al. [3], which represents the state of the art in dependency graph parsing in NLP.

Following the state of the art, we train stacked BiLSTMs and a biaffine classifier to infer the latent dependency graph connecting the words of a sentence. We find that, when not normalizing the scores produced by the biaffine transformation, model performance drops in terms of micro-averaged F_1-measure and attachment score [25]. In particular, increasing the amount of layers produces a normalization effect by reducing the variance of the output scores. Using score normalization, we find that in some cases similar or better performance can be obtained by reducing the amount of trained BiLSTM parameters by as much as 85%. A similar phenomenon is also observed with a GNN parser, where adding more layers is less effective when normalizing scores.

To ablate the BiLSTM architecture entirely, we also experiment with switching the BiLSTM layers with Transformer encoder layers. Furthermore, we briefly experiment with a 7B-parameter LLM, which constitutes a different architecture altogether.

Our contributions are thus three-fold. (*i*) We show the impact of normalizing the output of the biaffine scorer in relation to the architectural changes in a DP model. (*ii*) We show DP models can obtain better performance with substantially fewer trained parameters. (*iii*) We provide a new method that substantially improves scores and obtains state-of-the-art performance for semantic [22, 42] and syntactic [24, 44] dependency parsing.

The rest of the paper is structured as follows. Section 2 presents an overview of the literature concerning DP, with specific focus on how to infer the adjacency matrix of a dependency graph. Section 3 explains why layer depth can compensate for normalization. Section 4 provides an overview of the datasets, models, evaluation, and other experimental details. Section 5 shows how normalization mitigates overparameterization, both with and without multi-hop parsing with Graph Neural Networks (GNNs), and how it accelerates convergence. Finally, Section 6 draws conclusions and provides avenues for future research.

2 Background

In general, DP is an NLP task concerned with inferring the dependency graph of a sentence [30]. Depending on the nature of the nodes and relations of the graph, it can e.g., take on the name of semantic (SemDP) [3, 9, 16, 35] or syntactic (SynDP) [8, 14, 44]. In this work, we tackle both tasks, but focus more specifically on SemDP, which can be thought of as comprising two sub-tasks: Named Entity Recognition (NER) [23] and Relation Extraction (RE) [3, 16]. They can be either approached in a pipeline as separate objectives [6, 43, 46, 49], or by jointly training a single model on both sub-tasks [3, 4, 16, 45]. In the context of deep learning models trained end-to-end, three main paradigms are used: encoder-based models [3, 8, 9, 14, 16, 37, 38], decoder-only Transformers (LLMs) [4, 39, 47], and seq2seq encoder-decoder Transformers [18, 21, 26, 45].

In this work, we focus on encoder-style models, since they currently achieve the best performance on DP tasks [3, 16, 35] and are much more parameter efficient than LLM-based solutions. These models approach entity (node) prediction analogously to NER, while edges and relations are handled via MLP projection of node-pair features [17, 27, 28, 50], or via attention-based edge and relation inference [3, 9, 14, 16, 35]. Current state-of-the-art models, exemplified by [3], are based on the biaffine dependency parser introduced by [8], which uses stacked BiLSTM layers on top of embeddings from a pre-trained language model.

Other models also involve the use of GNNs for iterative adjacency matrix refinement [16]. In this case, intermediate representations and adjacency matrices are calculated by interleaved biaffine and GNN layers, before obtaining the final adjacency matrix. We experiment with such a setup using Graph Attention Networks (GATs) [5] to confirm the effectiveness of the normalization approach also for this kind of model.

3 Layer depth can compensate for normalization

In order to reduce the number of parameters used for biaffine scoring, we observe that the softmax function is sensitive to inputs with high variance: large values dominate the output ($e^{x \gg 1}$) and small values decay ($e^{x \ll 1}$). This causes a drop in the downstream task performance, due to some values dominating the probability outputs in the score matrix, as well as exploding and vanishing gradients. Typically, contemporary architectures based on [8] do not employ normalization, but are still able to perform well on DP tasks.

We explain this discrepancy using insights from the theory of implicit regularization [1], which states that, as layer depth increases, the effective rank of the weight matrices is reduced during gradient descent. Our claim is that a reduction in the rank of the weight matrices causes a corresponding decrease in the variance of the input to the softmax function.

Result 1 (Singular Value Dynamics under Gradient Descent [1]). *Minimization of a loss function $\mathcal{L}(W)$ with gradient descent using weight matrices $\mathbf{W}$ (assuming a small learning rate η and initialization close to the origin). An N-layer linear neural network leads the singular values of $\mathbf{W}$ to evolve in the number of iterations t by:*

$$\sigma_r(t+1) \leftarrow \sigma_r t - \eta \cdot \langle \nabla \mathcal{L}(W(t)), \mathbf{u}_r(t)\mathbf{v_r}^\top(t) \rangle \cdot N \cdot \sigma_r(t)^{2-2/N}$$

This implies that, as the number of layers increases, the rank of the weight matrices decreases, since the smallest singular values decay.

While Result 1 is stated for deep linear neural networks for the matrix factorization task, it has been validated empirically as applying to deep non-linear neural networks [32, 48]. We validate this finding for the biaffine scoring setting in Figure 1, which shows the inverse proportionality between the effective rank $\rho(W)$ [34] of a BiLSTM's weights and the number of its layers.

Claim 1 (Rank variance inequality). *Let $Y_r \in \mathbb{R}^m$ be a random vector that is the outcome of a linear transformation $\mathbf{y} = \mathbf{A}_r\mathbf{x}$, where the vector $\mathbf{x} \in \mathbb{R}^n$ is drawn from the random vector X and $\mathbf{A}_r \in \mathbb{R}^{m \times n}$ denotes a rank r approximation of a fixed matrix $\mathbf{A}$. The variance of Y_r is smaller than the variance of Y_{r+1} as measured by the Frobenius norm $||Var(Y_r)||_F \leq ||Var(Y_{r+1})||_F$.*

Proof. The variance of a random vector X is represented by the positive semi-definite (PSD) covariance matrix $\mathbf{K_{xx}} \in \mathbb{R}^{n \times n}$.

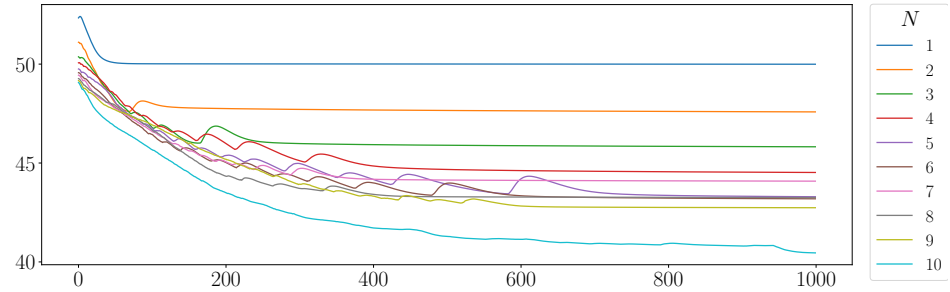


Figure 1: Effective rank $\rho(W)$ reduces over training epochs as we increase N of BiLSTM layers.

Applying the variance of X to the linear transformation, we get $\text{Var}(Y) = \mathbf{A_r K_{xx} A_r}^\top \in \mathbb{R}^{m \times m}$.

Since $\mathbf{K_{xx}}$ is PSD, so is each term in $\mathbf{x_i}^\top \mathbf{K_{xx}} \mathbf{x_i}$, and so the matrix $\mathbf{A_r K_{xx} A_r}^\top$ is also PSD for each rank-r approximation $\mathbf{A_r} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$.

This means that increasing the rank from $r-1$ to r adds a new PSD component Δ_r:

$$\mathbf{A_r K_{xx} A_r}^\top = \mathbf{A_{r-1} K_{xx} A_{r-1}}^\top + \Delta_r$$

Since the Frobenius norm is given by $\sqrt{\sum_i^m \sum_j^n |a_{ij}|^2}$, then we must have:

$$\begin{aligned} ||\mathbf{A_r K_{xx} A_r}^\top||_F &= ||\mathbf{A_{r-1} K_{xx} A_{r-1}}^\top + \Delta_r||_F \\ &\geq ||\mathbf{A_{r-1} K_{xx} A_{r-1}}^\top||_F \\ &= ||\mathbf{A_{r-2} K_{xx} A_{r-2}}^\top + \Delta_{r-1}||_F \\ &\dots \\ &\geq ||\mathbf{A_1 K_{xx} A_1}^\top||_F \end{aligned}$$

Which shows that, as the rank decreases, so does the variance of the linear transformation.

□

The implication of Claim 1 is that the pre-softmax score matrix from a shallow network without normalization will have a higher variance than a deeper network because of Result 1. Based on this finding, we propose to remove BiLSTM layers and use normalization —rather than having greater layer depth— in order to develop a more parameter-efficient method.

4 Experimental setting

4.1 Data

To train and evaluate our models, we use four SemDP datasets,[1] along with the 2.2 version of the Universal Dependencies English EWT treebank (enEWT) [24] and SciDTB [44].

As regards SemDP, **ADE** [12] is a medical-domain dataset comprising reports of drug adverse-effect reactions. Each disease is associated to a drug which caused it through the only type of relation present, i.e. "adverseEffect." **CoNLL04** [33] contains news texts and is annotated with the classic entities $e_i \in$ {per, org, loc} and relations $r_j \in$ {workFor, kill, orgBasedIn, liveIn, locIn}. ADE and CoNLL04 are characterized by relations which are not complex enough to form connected graphs of considerable size. **SciERC** [22] is a dataset compiled from sentences extracted from literature in the domain of artificial intelligence. It is arguably the most challenging dataset used in this study, since most of the entities in the validation and testing partitions are not assigned an entity class. This makes it hard to train tag embeddings that can help to infer edges. **ERFGC** [42] is a dataset comprising "flow graphs" parsed from culinary recipes. The semantic dependency graphs of these recipes are directed and acyclic, with a single root. Note that, differently from [3], and as advised directly by [42] in personal correspondence, we ignore the "-" edge labels present in the corpus.

[1]We neglect ACE04 and ACE05, two popular datasets for relation extraction, due to their prohibitive cost.

Table 1: Samples per partition and entity/relation classes for the datasets used in this paper.

Dataset (Train / Dev / Test) **Entities**	**Relations**
ADE (2,563 / 854 / 300) [12]	
disease, drug	adverseEffect
CoNLL04 (922 / 231 / 288) [33]	
organization, person, location	kill, locatedIn, workFor, orgBasedIn, liveIn
SciERC (1,366 / 187 / 397) [22]	
generic, material, method, metric, otherSciTerm, task	usedFor, featureOf, hyponymOf, evaluateFor, partOf, compare, conjunction
ERFGC (242 / 29 / 29) [42]	
food, tool, duration, quantity, actionByChef, discontAction, actionByFood, actionByTool, foodState, toolState	agent, target, indirectObject, toolComplement, foodComplement, foodEq, foodPartOf, foodSet, toolEq, toolPartOf, actionEq, timingHeadVerb, other
enEWT (10,098 / 1,431 / 1,427) [42]	
xPOS tags	UD relations
SciDTB (2,567 / 814 / 817) [44]	
xPOS tags	UD relations

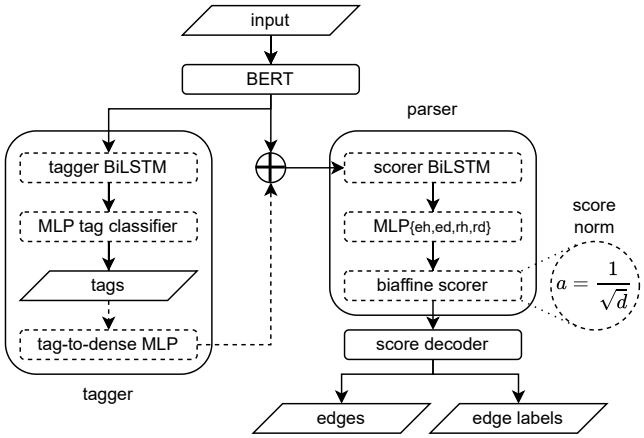


Figure 2: Dependency parsing diagram. Dashed components are the targets of ablation experiments.

As regards SynDP, we use **enEWT** [24] to compare directly against [14]'s results. While many works evaluate SynDP on the Penn Treebank [29] it is closed access and prohibitively expensive. Instead, we use **SciDTB** [44], a discourse analysis dataset comprising 798 abstracts extracted from the ACL Anthology. It was processed for the syntax dependency parsing task using Stanza [31]. From these two datasets, we consider the xPOS tags (e.g. noun, preposition, determiner) and the syntactic dependencies (e.g. sentence root, object, adverbial modifier) to respectively be the entities and relations of the dependency graphs to infer.

For ADE, CoNLL04, and SciERC we use the splits provided in [4].[2] ERFGC is not available online; we obtained it by contacting the authors of [42]. Table 1 summarizes the statistics of the datasets used in this work and reports their entity and relation class annotations.

4.2 Model

We adopt the architecture of [3] in our work, schematized in Figure 2. It can be subdivided into four main components: encoder, tagger, parser, and decoder. The input is tokenized and passed through a BERT-like encoder, where token representations are averaged into $|\mathcal{V}|$ word-level features $\mathbf{x}_i \in \mathbb{R}^{d_f}$.[3] Optionally, additional features can be obtained by predicting the entity classes of each word with a tagger, composed by a single-layer BiLSTM ϕ, followed by a classifier:

$$\mathbf{h}_i^{tag} = \phi(\mathbf{x}_i), \quad \mathbf{h}_i^{tag} \in \mathbb{R}^{d_h}$$
$$\mathbf{y}_i^{tag} = \text{Softmax}(\text{MLP}^{tag}(\mathbf{h}_i^{tag})), \quad \mathbf{y}_i^{tag} \in \mathbb{R}^{|T|}$$

where T is the set of word tag classes. The tagger's predictions are then converted into one-hot vectors and projected into dense representations by another MLP, such that $\mathbf{e}_i^{tag} = \text{MLP}^{emb}(\mathbf{1}_T(\mathbf{y}_i^{tag}))$. These new tag embeddings are concatenated with the original BERT output and sent to the parser.

[2] https://drive.google.com/drive/folders/1vVKJIUzK4hIipfdEGmS0CCoFmUmZwOQV
[3] Using token-level representations resulted in much lower performance in preliminary experiments.

In the parser, an optional N-layered BiLSTM ψ produces new representations $\mathbf{h}_i = \psi(\mathbf{e}_i^{tag} \oplus \mathbf{x}_i)$, which are then projected into four different representations:

$$\mathbf{e}_i^h = \text{MLP}^{(edge-head)}(\mathbf{h}_i), \ \mathbf{e}_i^d = \text{MLP}^{(edge-dept)}(\mathbf{h}_i)$$
$$\mathbf{r}_i^h = \text{MLP}^{(rel-dept)}(\mathbf{h}_i), \ \mathbf{r}_i^d = \text{MLP}^{(rel-head)}(\mathbf{h}_i)$$

The edge scores s_i^{edge} and relation scores s_i^{rel} are then calculated with the biaffine function f:

$$f(\mathbf{x}_1, \mathbf{x}_2; W) = \mathbf{x}_1^\top W \mathbf{x}_2 + \mathbf{x}_1^\top \mathbf{b}$$
$$s_i^{edge} = f^{(edge)}(\mathbf{e}_i^h, \mathbf{e}_i^d; W_e), \quad W_e \in \mathbb{R}^{d \times 1 \times d}$$
$$s_i^{rel} = f^{(rel)}(\mathbf{r}_i^h, \mathbf{r}_i^d; W_r), \quad W_r \in \mathbb{R}^{d \times |R| \times d}$$

where R is the set of relation classes, i.e. the possible labels applied to an edge.

Optionally, we experiment with the addition of $L_{\text{GNN}} \in \{0, 1, 2, 3\}$ GNN layers upstream of the final biaffine layer. Each layer is composed of a biaffine layer predicting an adjacency matrix based on the MLP outputs, sparsified to only keep the top k edge scores for each node. Each MLP output is then passed through a dedicated GAT layer [5] along with the sparse adjacency matrix:

$$\mathbf{e}_i^{l+1} = \sigma_1\left(\mathbf{e}_i^l, \sigma_2\left(\sum_{j \in \mathcal{N}_i}^{k} \alpha_{ij} \cdot W \mathbf{e}_j^l\right)\right)$$
$$\alpha_{ij} = \text{Softmax}_j(s(\mathbf{e}_i^l, \mathbf{e}_j^l))$$
$$s(\mathbf{e}_i^l, \mathbf{e}_j^l) = \mathbf{a}^\top \text{LeakyReLU}\left(W \cdot \left[\mathbf{e}_i \| \mathbf{e}_j\right]\right)$$

where σ_1, σ_2 are non-linearities, $\|$ is the concatenation operation, and $\mathcal{N}$ is the neighborhood of the i-th node.

Finally, in the decoder, the edge scores are used in conjunction with the relation representations $\mathbf{r}_i^h$ and $\mathbf{r}_i^d$ to obtain the final predictions. During training, we do greedy decoding, while during inference, we use Chu-Liu/Edmonds' maximum spanning tree (MST) algorithm [10] to ensure the predictions are well-formed trees. This is especially useful with big dependency graphs, since greedy decoding is more likely to produce invalid trees as size increases. When doing greedy decoding, an edge index (i.e. an adjacency matrix) $a_i = \arg\max_j s_{ij}^{edge}$ is produced by taking the argmax of the attention scores s_i^{edge} across the last dimension. The edge index is then used to select which head relation representations r_i^h to use to calculate the relation scores $s_i^{rel} = f(\mathbf{r}_i^h, \mathbf{r}_i^d; W), \quad W \in \mathbb{R}^{d \times |R| \times d}$. The relations are then predicted as $r_i = \arg\max_j s_{ij}^{rel}$. When using MST decoding, edge and relation scores are combined into a single energy matrix where each entry represents the score of a specific head-dependent pair with its most likely relation type. This energy matrix is then used in the MST algorithm, producing trees with a single root and no cycles. For all experiments, following [3], prior to energy calculation, edge scores and relation scores are scaled so that low values are squished and high values are increased, making the log softmax produce a hard adjacency matrix.

The model is trained end-to-end jointly on the entity, edge, and relation classification objectives:

$$\mathcal{L}_{tag} = -\frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|}\sum_{t=1}^{|T|} y_{i,t}^{tag}\,\log p\big(y_{i,t}^{tag}\big)$$

$$\mathcal{L}_{edge} = -\sum_{i,j=1}^{|\mathcal{V}|}\log p\big(y_{i,j}^{edge}=1\big)$$

$$\mathcal{L}_{rel} = -\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|}\mathbb{1}\big(y_{i,j}^{edge}=1\big)\sum_{\ell=1}^{|R|} y_{i,j,\ell}^{rel}\,\log p\big(y_{i,j,\ell}^{rel}\big)$$

$$\mathcal{L} = \lambda_1\,\mathcal{L}_{tag} + \lambda_2\big(\mathcal{L}_{edge} + \mathcal{L}_{rel}\big)$$

Losses are calculated based on the gold tags, edges, and relations. We set $\lambda_1 = 0.1$ and $\lambda_2 = 1$ as hyperparameters because the tagging task is much simpler than predicting the edges, since the same top performance is always achieved regardless of any other selected architecture hyperparameters. For the GNN setup, a separate loss is calculated for each biaffine layer, as in [14]. Following the usual approach for SynDP [8, 14, 16], when training on enEWT and SciDTB we use an oracle, the gold tags, and do not predict the POS tags ourselves. Since in this case we only focus on training the edge and relation classification tasks, we set $\lambda_1 = 0$.

To highlight the comparative parameter efficiency of the graph-based parsers, in Appendix B.9 we also experiment with a 7B-parameter LLM. Specifically, we pick Mistral-7B-Instruct-v0.3 [15],[4] because it has shown to achieve the best performance among models of similar size on ADE, CoNLL04, and SciERC [11].

4.3 Hyperparameters

We experiment with a range of hyperparameters for the encoder, tagger, and parser, as listed in Table 2. We use BERT$_{base}$ [7] as our pre-trained encoder, which we keep frozen throughout the whole training run in our main setting. As regards the tagger, we set $L_\phi = 1$ and $h_\phi = 100$, as in [3], with weights initialized with a Xavier uniform distribution. In Appendix B.1, we ablate the ϕ BiLSTM and the use of the tag embeddings $\mathbf{e}_i^{tag}$ to assess their impact on overall performance. Finally, with relation to the parser, for all hyperparameter combinations of $\{L_\psi, h_\psi, d_{\text{MLP}}\}$, we run our experiments by initializing its weights with a Xavier uniform distribution ($I_{par} = \mathcal{U}$) and no LayerNorm LN_ψ. In Appendices B.2 to B.7, we conduct ablations for Table 2's parser hyperparameters.

We train our models for $2k$ steps and evaluate on the development partition of each dataset every 100 steps for ADE, CoNLL04, SciERC, and ERFGC. For enEWT and SciDTB, we train for $5k$ steps with $1k$-step validation intervals to make our results more comparable with the start of the art [14]. We apply early stopping at 30% of the total steps without improvement and choose the best model based on top performance on the development split. In Appendix B.8, we extend the training to $10k$ steps and fully fine-tune a variety of small and large pre-trained language models to show time-wise test performance trends more clearly. In the case of the GNN experiment (Appendix B.6), we also increase the steps to $10k$, since the differences in performance become clearer later on in the training. We also train for $10k$ steps when using Transformer encoder blocks in place of BiLSTM layers (Appendix B.7), since the learning rate is lower and performance convergence is slower. Unless stated otherwise, we set the learning rate at $\eta = 1 \times 10^{-3}$ when the encoder is kept frozen. In all settings, including ablations, we use AdamW [19] as the optimizer and a batch size of 8.

[4]https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3

Table 2: Hyperparameter ranges tested in our experiments. ψ = Parser BiLSTM. f = biaffine layer.

Comp.	Hyperparameter	Values
Encoder	Freeze BERT	$\nabla_{\text{BERT}} \in \{\checkmark, \times\}$
Tagger	Tagger BiLSTM ϕ	$\phi \in \{\checkmark, \times\}$
	Concat. tag embeds.	$\mathbf{e}_i^{tag} \in \{\checkmark, \times\}$
Parser	ψ num. layers	$L_\psi \in \{0, 1, 2, 3\}$
	ψ hidden dim.	$h_\psi \in \{100, 200, 300, 400\}$
	$\text{MLP}^{(edge)}$ output dim.	$d_{\text{MLP}} \in \{100, 300, 500\}$
	ψ LayerNorm	$\text{LN}_\psi \in \{\checkmark, \times\}$
	$\text{MLP}^{(edge)}$ and $f^{(edge)}$ init.	$I_{\text{par}} \in \{\mathcal{U}, \mathcal{N}\}$
	GNN layers	$L_{\text{GNN}} \in \{0, 1, 2, 3\}$
	top-k edges	$k \in \{1, 4\}$
	Score scaling	$a \in \{1, \frac{1}{\sqrt{d}}\}$
	Parser type	$\psi \in \{\text{BiLSTM, Transf.}\}$

4.4 Evaluation

Following [3, 9, 16], we measure tagging and parsing performance on ADE, CoNLL04, SciERC, and ERFGC in terms of micro-F_1. In addition, for enEWT and SciDTB we use unlabeled (UAS) and labeled (LAS) attachment score [25]. To corroborate the validity of our results, we train and evaluate each setting, including ablations, with five random seeds. We report mean and standard deviation for the F_1, UAS, and LAS metrics, averaged over the five runs. For brevity, the performance on the tagging and unlabeled edge prediction tasks are reported in Appendix A. To test the significance of our results, we use the one-tailed Wilcoxon signed-rank test [40].

5 Results and discussion

Table 3 shows the results of our experiments, with the first row using the same hyperparameters as [3] ($h_\psi = 400, d_{\text{MLP}} = 500$). We also use these hyperparameters for enEWT and SciDTB, since our ablation study only concerns SemDP due to SynDP being less challenging. The lower half uses the best combinations for ADE (200, 100), CoNLL04 (400, 300), SciERC (300, 300), and ERFGC (400, 300), chosen based on mean performance across these four datasets.

Overall, ***normalizing biaffine scores provides an evident performance boost at all BiLSTM depths.*** In particular, for ERFGC we beat the state-of-the-art performance achieved by [3]. Most of the performance gain is obtained by adding the first layer, with

Table 3: Micro-F_1 (SemDP) and LAS (SynDP) for the labeled edge prediction task. Best in bold.

Model	a	L_ψ	ADE	CoNLL04	SciERC	ERFGC	enEWT	SciDTB
[3]	1	3	0.653±0.018	0.566±0.019	0.257±0.024	0.701±0.009	0.804 ±0.006	0.915 ±0.002
Ours	1	0	0.541 ±0.021	0.399 ±0.024	0.147 ±0.049	0.548 ±0.010	0.559 ±0.005	0.729 ±0.004
		1	0.657 ±0.011	0.556 ±0.021	0.282 ±0.009	0.676 ±0.010	0.771 ±0.006	0.892 ±0.002
		2	0.667 ±0.011	0.573 ±0.025	0.273 ±0.010	0.694 ±0.010	0.796 ±0.006	0.910 ±0.002
		3	0.662 ±0.027	0.562 ±0.021	0.299 ±0.023	0.705 ±0.011	0.804 ±0.006	0.915 ±0.002
	$\frac{1}{\sqrt{d}}$	0	0.567 ±0.014	0.438 ±0.033	0.181 ±0.027	0.612 ±0.008	0.646 ±0.002	0.796 ±0.002
		1	0.668 ±0.017	0.597 ±0.015	0.299 ±0.019	0.692 ±0.009	0.789 ±0.003	0.904 ±0.002
		2	0.676 ±0.019	0.596 ±0.014	0.312 ±0.011	0.699 ±0.009	0.805 ±0.003	0.916 ±0.002
		3	**0.686** ±0.025	**0.602** ±0.017	**0.320** ±0.013	**0.708** ±0.008	**0.807** ±0.005	**0.919** ±0.001

additional ones yielding diminishing returns. In other words, the performance boost provided by score normalization is highest in the absence of the implicit normalization provided by extra parameters ($L_\psi > 0$). In general, the top performance obtained by using three BiLSTM layers and no score normalization can be matched or surpassed ***with a single BiLSTM layer, when using score normalization.*** Taking into account the lower values for h_ψ and d_{MLP}, this represents a reduction in trained parameters of up to 85%.

As laid out in Section 3, score variance tends to decay with deeper BiLSTM stacks. This in turn produces a converging trend as L_ψ increases. When looking at Figure 3, this is especially evident for ERFGC, enEWT, and SciDTB, for which the beneficial effect of score normalization shrinks smoothly with higher values of L_ψ. Although the same cannot be said for ADE, CoNLL04, and SciERC, the performance boost is still statistically significant across all layer depths ($p < 0.01$).

As regards SciERC, normalizing scores without any BiLSTM layers ($L_\psi = 0$) produces a 23% increase in performance with a reduction in standard deviation. SciERC arguably has the hardest dependency graphs to parse, due to the little overlap between training and testing entities, which makes it difficult to leverage tag embeddings. In addition, it is characterized by complex semantic dependencies. Thus, we argue normalizing biaffine scores is especially important when dealing with hard tasks. In particular, normalizing scores helps mitigate the higher variance of the model's predictions for this challenging dataset. Conversely, the lack of score normalization exacerbates already uncertain predictions. For SciERC, this makes the trend observed in Figure 3 more unstable. Indeed, the performance first drops at $L_\psi = 2$ and then increases once again at $L_\psi = 3$, which does not happen for the other datasets.

Compared to existing approaches which do not scale the biaffine scores, fewer parameters can thus be trained to obtain similar performance. Therefore, our results indicate that parsers using raw biaffine scores are likely overparameterized, compared to the performance they could achieve by using score normalization. In addition, score normalization increases sample efficiency by accelerating convergence. Figure 4 displays the performance on SciERC's test set for four models during full fine-tuning. As the figure shows, the speedup in convergence is statistically significant when normalizing scores. This shows how score normalization can be beneficial even when fully fine-tuning models with ~10^8 parameters, given a hard task which forces the model to make uncertain predictions.

Figure 5 shows the performance on SciDTB with respect to the use of normalization and the number of GNN layers inserted upstream of the biaffine scoring.[5] Top performance increases visibly with $a = 1/\sqrt{d}$, with the performance trends also being substantially more stable. Confirming the findings of [14], the addition of one or more GNN layers is beneficial, showing the usefulness of considering multi-hop dependency interactions. Specifically, using a single GNN layer provides the highest performance. This possibly indicates that going over 2-hop interactions may not be useful due to over-smoothing of the representations. As regards the number k of highest-score edges considered by the GAT layers, the best results are obtained by using the top $k = 4$ edges at each GNN layer. In addition, taking into account more likely edges is increasingly beneficial the more GNN layers are used. Thus, aggregating information from a bigger neighborhood starts to become desirable once the adjacency matrix is refined a sufficient number of times. Intuitively, with $k = 1$ choosing the wrong edge with $L_{\text{GNN}} = 1$ only creates minor noise by selecting wrong edges at 2-hop distance. However, at $k = 1$, propagating multiple times lowers performance because errors in edge selection lead to multiplied propagation of unwanted representation aggregations. Conversely, with a higher $k = 4$, the correct representations are more likely to be aggregated, which is more beneficial with higher numbers of layers. This is evident by looking at the performance with $k = 1$ at $L_{\text{GNN}} = 3$, which is similar or lower to the one with $L_{\text{GNN}} = 0$, depending on a. With $k = 4$, however, the performance at $L_{\text{GNN}} = 3$ is clearly higher than the one at at $L_{\text{GNN}} = 0$. Regarding the effect of a, once again normalization greatly reduces the variance in performance. This is particularly evident with incresing numbers of GNN layers, since the effect accumulates iteratively. Finally, this experiment shows that using a single BiLSTM layer provides much greater performance compared to using three GNN layers, despite both of these settings having ~$7M$ learnable parameters. This puts into question the effectiveness of [14]'s approach, when using modern architectures such as [3].

[5]Note that although the results at $L_{\text{GNN}} = 0$ are equivalent to those of Table 3 for $L_\psi = 0$, here we are simply reporting the maximum value of the curve, not the maximum test performance at the highest validation performance.

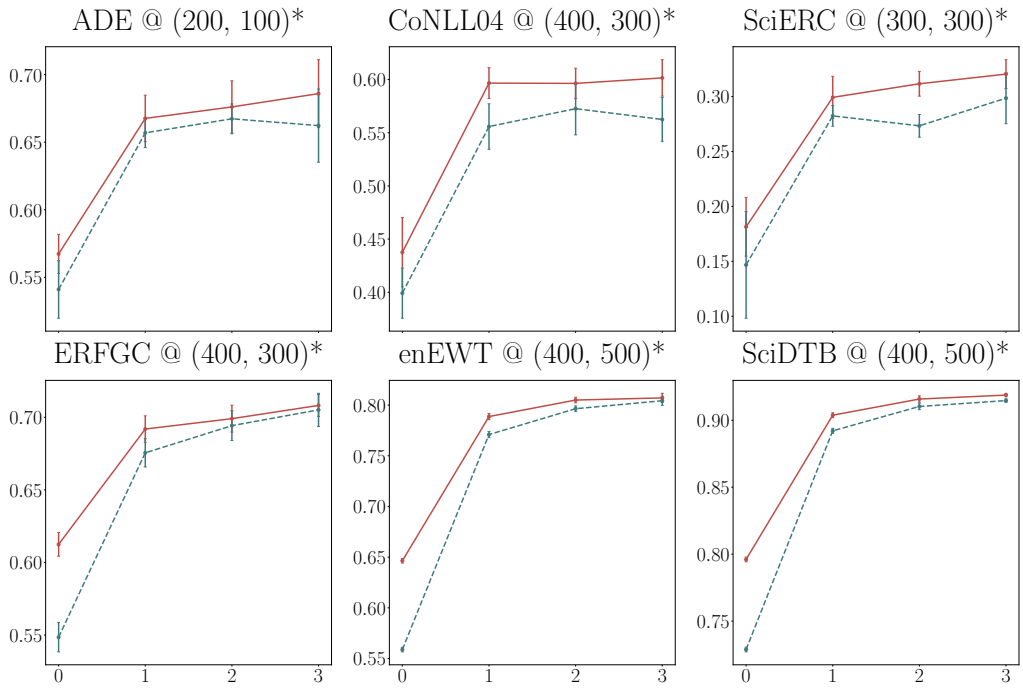


Figure 3: Micro-F_1 (SemDP) and LAS (SynDP) vs $L_\psi \in \{0, 1, 2, 3\}$ at $2k$ training steps. Red = norm; blue = raw. *Performance increase with normalization is statistically significant ($p < 0.01$).

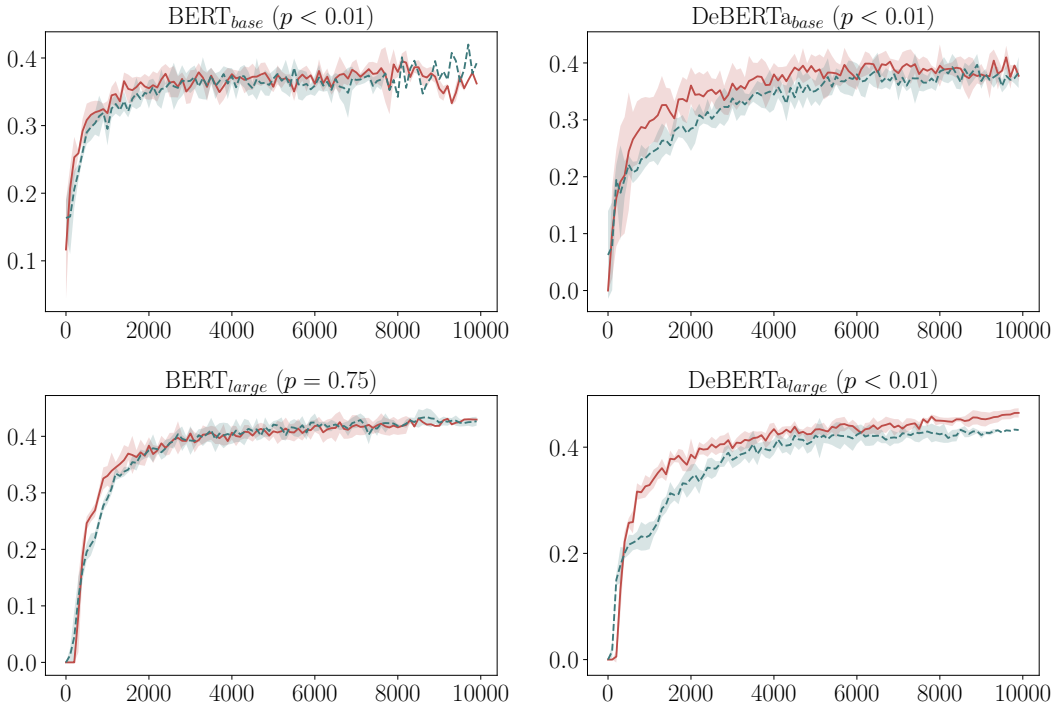


Figure 4: Test performance on SciERC (micro-averaged F_1-measure vs number of training steps). The p-values indicate greater performance with normalization (one-tailed Wilcoxon signed-rank test).

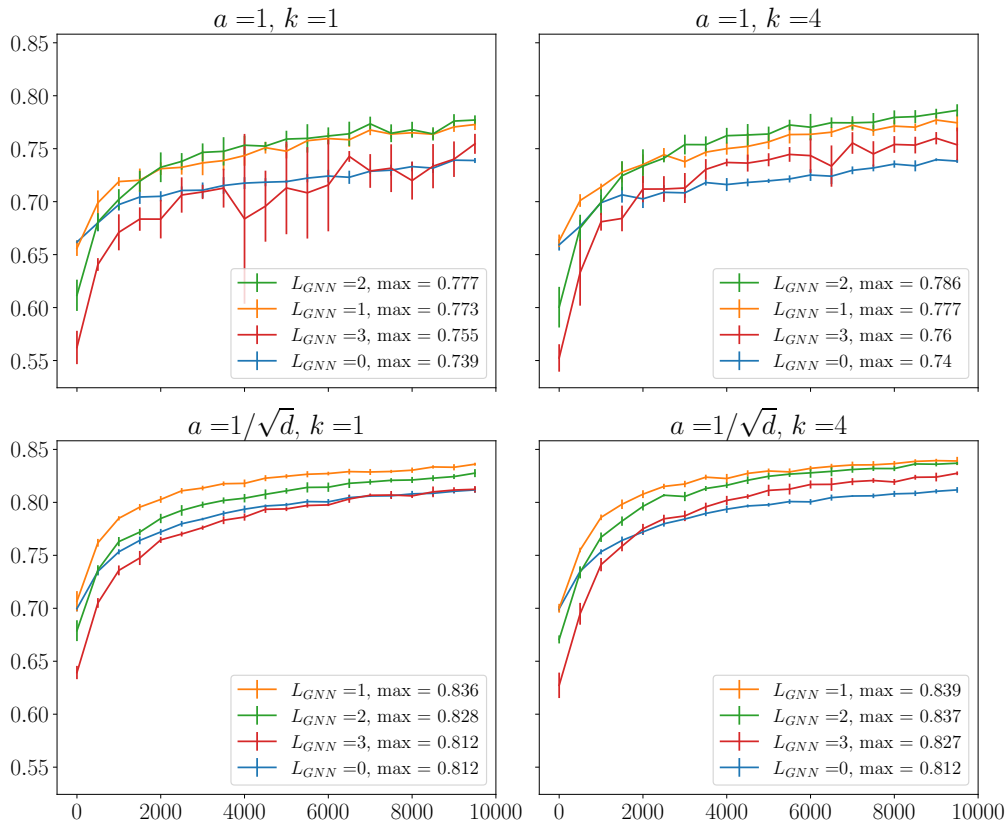


Figure 5: Test performance on SciDTB using GNN Layers (LAS vs number of training steps).

6 Conclusions

In this work, we explored the effect of scaling the scores produced by biaffine transformations when predicting the edges of a dependency graph. We have demonstrated, both theoretically and empirically, that the score variance produced by a lack of score scaling hurts model performance when predicting edges and relations. In addition, our theoretical work and experiments highlight a strong relationship between the number of trained layers and their intrinsic normalization effect.

Departing from a state-of-the-art architecture for semantic dependency parsing, we were able to improve its performance on both semantic and syntactic dependency parsing on six datasets. On ERFGC, a dataset of directed acyclic semantic dependency graphs compiled from culinary recipes, our approach allowed us to beat the state-of-the-art performance achieved by Bhatt et al. [3]. Moreover, our results showed that a single BiLSTM layer can be sufficient to match or surpass the results of state-of-the-art architectures with a decrease in trained parameters of up to 85%. In the case of SciERC, a challenging dataset for semantic dependency parsing, we found that the performance boost was particularly great when only training the biaffine scorer, without any BiLSTM layers. Moreover, for this challenging dataset, we find that scaling the predictions of the biaffine scorer can accelerate convergence speed even when fully fine-tuning models in the 100 to 400M parameter range. In addition, for three of the datasets we also observed that the performance obtained with normalized and raw scores converged smoothly as the number of trained layers increased. This supported our claim that stacking BiLSTM layers mainly serves the purpose of producing an implicit regularization, and that this effect can be obtained by normalizing the scores, without any extra parameters. We have also shown that using Transformer encoder layers instead of BiLSTM layers does not produce a performance boost, despite higher parameter counts. Furthermore, we have demonstrated that score normalization is also effective when using multi-hop GNN parser. Moreover, obtaining richer representations by using a single BiLSTM layer with a one-hop parser yields much greater performance compared to multi-hop parsing via multiple GAT layers. Finally, in order to have a comparison with models with very large parameter counts, we also experimented with a 7B-parameter LLM, showing its performance to be rather poor on complex graphs.

In the future, we aim to approach large scale graph inference tasks which have been limited by the lack of parameter efficient methods, such as long form discourse parsing tasks. As regards score scaling, we wish to verify whether its positive effects can carry over to other tasks. For example, we plan to extend our work to non-NLP tasks, such as molecular graph inference, using the QM9 dataset [41]. Further investigation is also warranted in model efficiency, given our findings, e.g. via pruning of model parameters.

References

[1] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. 2019. Implicit Regularization in Deep Matrix Factorization. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/c0c783b5fc0d7d808f1d14a6e9c8280d-Paper.pdf

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[3] Dhaivat J. Bhatt, Seyed Ahmad Abdollahpouri Hosseini, Federico Fancellu, and Afsaneh Fazly. 2024. End-to-end Parsing of Procedural Text into Flow Graphs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (Eds.). ELRA and ICCL, Torino, Italia, 5833–5842. https://aclanthology.org/2024.lrec-main.517

[4] Zhen Bi, Jing Chen, Yinuo Jiang, Feiyu Xiong, Wei Guo, Huajun Chen, and Ningyu Zhang. 2024. CodeKGC: Code Language Model for Generative Knowledge Graph Construction. doi:10.48550/arXiv.2304.09048 arXiv:2304.09048 [cs].

[5] Shaked Brody, Uri Alon, and Eran Yahav. 2022. How Attentive are Graph Attention Networks?. In *International Conference on Learning Representations*.

[6] Yee Seng Chan and Dan Roth. 2011. Exploiting syntactico-semantic structures for relation extraction. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*. 551–560.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. doi:10.18653/v1/N19-1423

[8] Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *International Conference on Learning Representations*. arXiv. http://arxiv.org/abs/1611.01734 arXiv:1611.01734 [cs].

[9] Timothy Dozat and Christopher D. Manning. 2018. Simpler but More Accurate Semantic Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 484–490. doi:10.18653/v1/P18-2077

[10] Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics* 71B, 4 (Oct. 1967), 233. doi:10.6028/jres.071B.032

[11] Gajo and Barrón-Cedeño. 2025. Natural vs Programming Language in LLM Knowledge Graph Construction. *Information Processing & Management* 62, 5 (2025), 104195. doi:10.1016/j.ipm.2025.104195

[12] Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of Biomedical Informatics* 45, 5 (2012), 885–892. doi:10.1016/j.jbi.2012.04.008 Text Mining and Natural Language Processing in Pharmacogenomics.

[13] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. http://arxiv.org/abs/2106.09685 arXiv:2106.09685 [cs].

[14] 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*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 2475–2485. doi:10.18653/v1/P19-1237

[15] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. doi:10.48550/arXiv.2310.06825 arXiv:2310.06825 [cs].

[16] Shu Jiang, Zuchao Li, Hai Zhao, and Weiping Ding. 2024. Entity-Relation Extraction as Full Shallow Semantic Dependency Parsing. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 32 (2024), 1088–1099. doi:10.1109/TASLP.2024.3350905

[17] Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics* 4 (July 2016), 313–327. doi:10.1162/tacl_a_00101 _eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00101/1567410/tacl_a_00101.pdf.

[18] Tianyu Liu, Yuchen Eleanor Jiang, Nicholas Monath, Ryan Cotterell, and Mrinmaya Sachan. 2022. Autoregressive Structured Prediction with Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 993–1005. doi:10.18653/v1/2022.findings-emnlp.70

[19] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. http://arxiv.org/abs/1711.05101 arXiv:1711.05101 [cs, math].

[20] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. http://arxiv.org/abs/1711.05101

[21] Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022. Unified Structure Generation for Universal Information Extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 5755–5772. doi:10.18653/v1/2022.acl-long.395

[22] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3219–3232. doi:10.18653/v1/D18-1360

[23] David Nadeau and Satoshi Sekine. 2009. A survey of named entity recognition and classification. In *Recognition, classification and use*, Satoshi Sekine and Elisabete Ranchhod (Eds.). John Benjamins Publishing Company, 3–28. doi:doi:10.1075/bct.19.03nad

[24] Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, and Lene Antonsen. 2018. Universal Dependencies 2.2. http://hdl.handle.net/11234/1-2837 LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[25] Joakim Nivre and Chiao-Ting Fang. 2017. Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. 86–95.

[26] Giovanni Paolini, Ben Athiwaratkun, Jason Krone, Jie Ma, Alessandro Achille, RISHITA ANUBHAI, Cicero Nogueira dos Santos, Bing Xiang, and Stefano Soatto. 2021. Structured Prediction as Translation between Augmented Natural Languages. In *International Conference on Learning Representations*. https://openreview.net/forum?id=US-TP-xnXI

[27] Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An Effective Neural Network Model for Graph-based Dependency Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong and Michael Strube (Eds.). Association for Computational Linguistics, Beijing, China, 313–322. doi:10.3115/v1/P15-1031

[28] Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep Multitask Learning for Semantic Dependency Parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 2037–2048. doi:10.18653/v1/P17-1186

[29] Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind Joshi, and Bonnie Webber. 2008. The Penn Discourse TreeBank 2.0.. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, and Daniel Tapias (Eds.). European Language Resources Association (ELRA), Marrakech, Morocco. https://aclanthology.org/L08-1093/

[30] Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. 2019. Universal dependency parsing from scratch. *arXiv preprint arXiv:1901.10457* (2019).

[31] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

[32] Noam Razin and Nadav Cohen. 2020. Implicit regularization in deep learning may not be explainable by norms. *Advances in neural information processing systems* 33 (2020), 21174–21187.

[33] Dan Roth and Wen-Tau Yih. 2004. A Linear Programming Formulation for Global Inference in Natural Language Tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*. Association for Computational Linguistics, Boston, Massachusetts, USA, 1–8. https://aclanthology.org/W04-2401

[34] Olivier Roy and Martin Vetterli. 2007. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*. IEEE, 606–610.

[35] Wei Tang, Benfeng Xu, Yuyue Zhao, Zhendong Mao, Yifeng Liu, Yong Liao, and Haiyong Xie. 2022. UniRel: Unified Representation and Interaction for Joint Relational Triple Extraction. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 7087–7099. doi:10.18653/v1/2022.emnlp-main.477

[36] 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*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/hash/

3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[37] David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. 2019. Entity, Relation, and Event Extraction with Contextualized Span Representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 5784–5789. doi:10.18653/v1/D19-1585

[38] Jue Wang and Wei Lu. 2020. Two are Better than One: Joint Entity and Relation Extraction with Table-Sequence Encoders. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 1706–1721. doi:10.18653/v1/2020.emnlp-main.133

[39] Xiang Wei, Xingyu Cui, Ning Cheng, Xiaobin Wang, Xin Zhang, Shen Huang, Pengjun Xie, Jinan Xu, Yufeng Chen, Meishan Zhang, Yong Jiang, and Wenjuan Han. 2024. ChatIE: Zero-Shot Information Extraction via Chatting with ChatGPT. http://arxiv.org/abs/2302.10205 arXiv:2302.10205 [cs].

[40] Robert F Woolson. 2005. Wilcoxon signed-rank test. *Encyclopedia of biostatistics* 8 (2005).

[41] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chem. Sci.* 9, 2 (2018), 513–530. doi:10.1039/C7SC02664A Publisher: The Royal Society of Chemistry.

[42] Yoko Yamakata, Shinsuke Mori, and John Carroll. 2020. English Recipe Flow Graph Corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association, Marseille, France, 5187–5194. https://aclanthology.org/2020.lrec-1.638

[43] Zhiheng Yan, Chong Zhang, Jinlan Fu, Qi Zhang, and Zhongyu Wei. 2021. A Partition Filter Network for Joint Entity and Relation Extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 185–197. doi:10.18653/v1/2021.emnlp-main.17

[44] An Yang and Sujian Li. 2018. SciDTB: Discourse Dependency TreeBank for Scientific Abstracts. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Melbourne, Australia, 444–449. doi:10.18653/v1/P18-2071

[45] Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. 2024. An autoregressive text-to-graph framework for joint entity and relation extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19477–19487. Issue: 17.

[46] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. 2003. Kernel methods for relation extraction. *Journal of machine learning research* 3, Feb (2003), 1083–1106.

[47] Bowen Zhang and Harold Soh. 2024. Extract, Define, Canonicalize: An LLM-based Framework for Knowledge Graph Construction. http://arxiv.org/abs/2404.03868 arXiv:2404.03868 [cs].

[48] Dan Zhao. 2022. Combining explicit and implicit regularization for efficient learning in deep networks. *Advances in Neural Information Processing Systems* 35 (2022), 3024–3038.

[49] Zexuan Zhong and Danqi Chen. 2021. A Frustratingly Easy Approach for Entity and Relation Extraction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 50–61. doi:10.18653/v1/2021.naacl-main.5

[50] Hao Zhu, Yankai Lin, Zhiyuan Liu, Jie Fu, Tat-Seng Chua, and Maosong Sun. 2019. Graph Neural Networks with Generated Parameters for Relation Extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 1331–1339. doi:10.18653/v1/P19-1128

A Full Results

Table 4 reports the results for the best hyperparameter combinations for all three tasks of predicting tags, edges, and relations of the semantic dependency graphs. Note that the first part of Table 4 is identical to the bottom of Table 3. Since we already discussed the performance for labeled edges in Section 5, we forgo discussing them again in this appendix.

As regards unlabeled edges, using score normalization improves mean performance for all datasets, similarly to relations. This is expected, as unlabeled edges directly influence the prediction of the edge labels, together with entity class prediction. As we have already observed for relations, the performance boost provided by normalization is particularly great for SciERC at $L_\psi = 0$.

It is also interesting to notice that for CoNLL04, the performance for the unlabeled edge prediction task is only slightly higher than that of the relations. In fact, as regards ADE, the opposite is true, with the labeled performance being seemingly higher. Due to the high variances, it would therefore seem that for these two datasets there is essentially no difference in difficulty between the labeled and unlabeled edge prediction tasks. In the case of ADE, this makes perfect sense, since there is only one possible relation. For CoNLL04, the amount of relation types is rather limited as well; therefore, it is sensible for the performance to also be similar in this case. Indeed, when looking at SciERC and ERFGC, the performance gap between labeled and unlabeled tasks is considerable. This is a strong indication that CoNLL04 is thus found somewhere in the middle, where the number of possible relations only slightly affects labeled performance.

As far as enEWT and SciDTB are concerned, the performance of edges (UAS) and relations (LAS) is also rather similar. Since in this case the predictions involve syntactic rather than semantic relations, the similar performance hints at relations being easier to predict in SynDP than in SemDP.

In the tagging task, very high standard deviation can be observed. In the case of ADE and CoNLL04, this could be caused by our choice of $\lambda_1 = 0.1$ being too low a value. Regarding SciERC, as already mentioned, most of the entities in the validation and test set are not found in the training set. This means it is not possible to train the model to recognize them, which results in the abysmal tagging performance reported in Table 4. This makes it challenging to train good tag embeddings which can help inform the edge and relation prediction tasks. Surprisingly, we notice that for CoNLL04 and ERFGC the tagging performance is also seemingly higher with score normalization. However, the overlap of the standard deviations is too high to be able to make any claims on the matter.

B Ablations

B.1 Tagger

Table 5 reports the results for the best values of h_ψ and d_{MLP} (chosen as described in Section 4), ablating over $\phi \in \{\checkmark, \times\}$ and $\mathbf{e}_i^{tag} \in \{\checkmark, \times\}$. Note that the first quarter of the table is equivalent to Table 3.

Using both the tagger BiLSTM and tag embeddings on average produces the best results. This is sensible, since a better tagger produces better tag embeddings, which in turn help inform the edge and relation classification tasks. The model in this case obtains the best performance on ADE, CoNLL04, and ERFGC. Its top performance for SciERC (0.320) is also close to the overall peak performance (0.324), obtained without using the tagger BiLSTM. On average, the mean performance is considerably higher when combining the positive contributions of the BiLSTM and the tag embeddings.

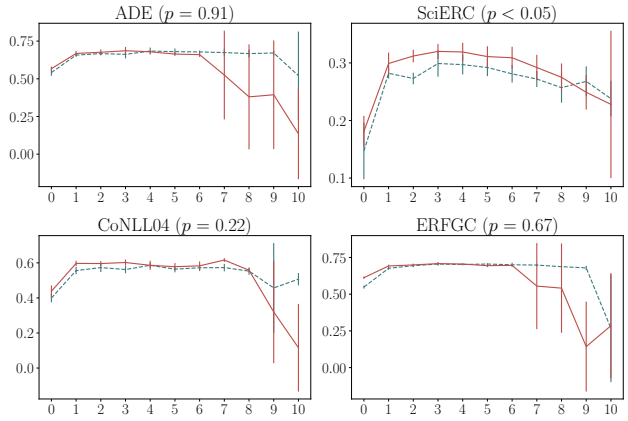


Figure 6: Performance in terms of F_1-measure vs $L_\psi \in \{0, \ldots, 10\}$ on ADE, CoNLL04, SciERC, and ERFGC at $2k$ training steps. As hyperparameters, we use the best values (h_ψ, d_{MLP}) with $\phi = \checkmark$ and $\mathbf{e}_i^{tag} = \checkmark$. Red = norm; blue = raw. The p-values indicate that the performance gap between using normalized and raw scores is statistically significant (one-tailed Wilcoxon signed-rank test).

B.2 Additional BiLSTM layers

As shown in Figure 6, performance is initially greater when normalizing scores, with $L_\psi \in \{0, 1, 2, 3, 4, 5\}$. In this range, an increasing amount of layers reduces the amount of benefit obtained by using normalization. However, at $L_\psi > 5$, for ADE, CoNNL04, and ERFGC, performance drops suddenly and rapidly when using biaffine score normalization. In the case of SciERC, the drop is more gentle, with the performance for the normalized and raw biaffine scores eventually converging, albeit with very high standard deviation at $L_\psi = 10$.

As observed in Section 5, adding trainable layers seemingly allows the parameters to scale the variance of the scores to compensate for the lack of explicit normalization. As reported in Appendix B.8, a similar behavior can be observed with fully fine-tuned models, where normalization yields diminishing benefits the more parameters we train. This behavior points to a trade-off between the use of our technique and the amount of stacked BiLSTM layers, at a given amount of training steps.

B.3 MLP output dimension

As shown in Table 6, without normalization, increasing the output dimension of the two MLPs responsible for projecting edge representation projections leads to a decrease in performance. This

Table 4: Micro-averaged test F_1 performance on all tasks ($\phi = \checkmark$, $\mathbf{e}_i^{tag} = \checkmark$). Best in bold.

Metric	a	L_ψ	ADE	CoNLL04	SciERC	ERFGC	enEWT	SciDTB
$(h_\psi, d_{\text{MLP}}) =$			(200, 100)	(400, 300)	(300, 300)	(400, 300)	(400, 500)	(400, 500)
rels.	1	0	0.541 ±0.021	0.399 ±0.024	0.147 ±0.049	0.548 ±0.010	0.559 ±0.005	0.729 ±0.004
		1	0.657 ±0.011	0.556 ±0.021	0.282 ±0.009	0.676 ±0.010	0.771 ±0.006	0.892 ±0.002
		2	0.667 ±0.011	0.573 ±0.025	0.273 ±0.010	0.694 ±0.010	0.796 ±0.006	0.910 ±0.002
		3	0.662 ±0.027	0.562 ±0.021	0.299 ±0.023	0.705 ±0.011	0.804 ±0.006	0.915 ±0.002
	$\frac{1}{\sqrt{d}}$	0	0.567 ±0.014	0.438 ±0.033	0.181 ±0.027	0.612 ±0.008	0.646 ±0.002	0.796 ±0.002
		1	0.668 ±0.017	0.597 ±0.015	0.299 ±0.019	0.692 ±0.009	0.789 ±0.003	0.904 ±0.002
		2	0.676 ±0.019	0.596 ±0.014	0.312 ±0.011	0.699 ±0.009	0.805 ±0.003	0.916 ±0.002
		3	**0.686** ±0.025	**0.602** ±0.017	**0.320** ±0.013	**0.708** ±0.008	**0.807** ±0.005	**0.919** ±0.001
edges	1	0	0.536 ±0.009	0.415 ±0.020	0.173 ±0.050	0.601 ±0.013	0.589 ±0.006	0.745 ±0.005
		1	0.652 ±0.009	0.566 ±0.022	0.321 ±0.009	0.744 ±0.013	0.793 ±0.006	0.901 ±0.002
		2	0.657 ±0.021	0.586 ±0.017	0.322 ±0.017	0.769 ±0.014	0.819 ±0.006	0.919 ±0.002
		3	0.649 ±0.027	0.578 ±0.011	0.355 ±0.028	0.782 ±0.007	0.827 ±0.006	0.924 ±0.002
	$\frac{1}{\sqrt{d}}$	0	0.549 ±0.014	0.453 ±0.032	0.203 ±0.030	0.674 ±0.007	0.682 ±0.003	0.815 ±0.001
		1	0.654 ±0.020	0.598 ±0.021	0.351 ±0.019	0.762 ±0.009	0.810 ±0.003	0.913 ±0.002
		2	0.660 ±0.015	0.600 ±0.008	0.367 ±0.015	0.775 ±0.008	0.827 ±0.003	0.925 ±0.002
		3	**0.666** ±0.028	**0.610** ±0.022	**0.377** ±0.021	**0.786** ±0.004	**0.829** ±0.004	**0.928** ±0.002
tags	1	0	0.615 ±0.115	0.285 ±0.150	0.014 ±0.015	0.662 ±0.073		
		1	0.665 ±0.147	0.671 ±0.018	0.049 ±0.014	0.794 ±0.057		
		2	0.746 ±0.011	0.648 ±0.074	0.060 ±0.012	0.838 ±0.016		
		3	**0.768** ±0.022	0.669 ±0.033	**0.062** ±0.005	0.853 ±0.013		
	$\frac{1}{\sqrt{d}}$	0	0.604 ±0.134	0.464 ±0.115	0.046 ±0.005	0.735 ±0.066		
		1	0.734 ±0.048	0.702 ±0.031	0.058 ±0.014	0.857 ±0.010		
		2	0.724 ±0.064	0.688 ±0.080	0.060 ±0.007	**0.864** ±0.018		
		3	0.762 ±0.017	**0.690** ±0.062	0.057 ±0.012	0.850 ±0.022		

is in contrast with the behavior observed when using normalization, where performance is essentially independent from the output dimension. This is in line with our claim that higher variance in the edge scores causes lower performance. Normalizing the scores assuages the variance, which makes performance stable even at high output dimensions. Once again, these results show the relationship between score variance and performance, with equivalent performance being achievable with fewer parameters.

B.4 BiLSTM hidden size

As Table 7 shows, the hidden size of the BiLSTMs does not have a visible effect on performance. This is especially the case when applying score normalization, which produces smaller standard deviations. As a result, not only do models perform better with biaffine score normalization, but performance is also less dependent on the hidden size of the BiLSTM encoders. This supports our claim that score normalization is a useful technique to obtain the same performance with lower parameter count, since the models tend to perform comparably, despite the lower BiLSTM hidden sizes.

B.5 LayerNorm and parameter initialization

In this section, we analyze the effects of using a Xavier normal distribution ($I_{par} = \mathcal{N}$)[6] for the weights of the two projections $\text{MLP}^{(edge-head)}$ and $\text{MLP}^{(edge-dept)}$, along with the edge biaffine layer $f^{(edge)}(\cdot\,; W_e)$. We also apply a LayerNorm function LN_ψ [2] for each of the BiLSTM layers.

As reported in Table 8, on average the best results are obtained with the base setting, i.e. with $I_{par} = \mathcal{U}$ and $\text{LN}_\psi = \times$. Using LayerNorm can provide a performance boost for some datasets, especially with uniform initialization. However, it makes performance unstable for some. As a matter of fact, when using LayerNorm with three BiLSTM layers on CoNLL04 and SciERC, the model often breaks and is not able to converge. Based on average performance, then, the best models are obtained by scaling biaffine scores ($a = 1/\sqrt{d}$) and initializing the parser with a uniform weight distribution, without any LayerNorm in between the stacked BiLSTM layers.

Table 5: Tagger ablation results: performance for the best hyperparameters (h_ψ, d_{MLP}). Best in bold, second-best underlined.

a	L_ψ	ϕ	$\mathbf{e}_i^{tag}$	ADE	CoNLL04	SciERC	ERFGC	Mean
(h_ψ, d_{MLP}) =				(200, 100)	(400, 300)	(300, 300)	(400, 300)	
1	0	■	■	0.541 ±0.021	0.399 ±0.024	0.147 ±0.049	0.548 ±0.010	0.515
	1	■	■	0.657 ±0.011	0.556 ±0.021	0.282 ±0.009	0.676 ±0.010	
	2	■	■	0.667 ±0.011	0.573 ±0.025	0.273 ±0.010	0.694 ±0.010	
	3	■	■	0.662 ±0.027	0.562 ±0.021	0.299 ±0.023	0.705 ±0.011	
$\frac{1}{\sqrt{d}}$	0	■	■	0.567 ±0.014	0.438 ±0.033	0.181 ±0.027	0.612 ±0.008	**0.541**
	1	■	■	0.668 ±0.017	0.597 ±0.015	0.299 ±0.019	0.692 ±0.009	
	2	■	■	0.676 ±0.019	0.596 ±0.014	0.312 ±0.011	0.699 ±0.009	
	3	■	■	**0.686** ±0.025	**0.602** ±0.017	0.320 ±0.013	**0.708** ±0.008	
1	0	■	□	0.543 ±0.013	0.402 ±0.023	0.162 ±0.019	0.554 ±0.008	0.517
	1	■	□	0.674 ±0.011	0.527 ±0.026	0.275 ±0.013	0.677 ±0.005	
	2	■	□	0.672 ±0.019	0.575 ±0.009	0.293 ±0.012	0.689 ±0.011	
	3	■	□	0.657 ±0.027	0.583 ±0.011	0.301 ±0.012	0.697 ±0.007	
$\frac{1}{\sqrt{d}}$	0	■	□	0.563 ±0.013	0.443 ±0.019	0.188 ±0.006	0.612 ±0.003	0.534
	1	■	□	0.653 ±0.023	0.565 ±0.027	0.299 ±0.020	0.687 ±0.006	
	2	■	□	0.672 ±0.017	0.592 ±0.020	0.307 ±0.008	0.702 ±0.005	
	3	■	□	0.661 ±0.022	0.593 ±0.017	0.305 ±0.014	**0.708** ±0.007	
1	0	□	■	0.550 ±0.026	0.387 ±0.030	0.157 ±0.012	0.553 ±0.006	0.514
	1	□	■	0.667 ±0.022	0.532 ±0.036	0.273 ±0.013	0.673 ±0.006	
	2	□	■	0.665 ±0.021	0.565 ±0.024	0.278 ±0.027	0.694 ±0.013	
	3	□	■	0.676 ±0.021	0.558 ±0.051	0.288 ±0.012	0.706 ±0.006	
$\frac{1}{\sqrt{d}}$	0	□	■	0.578 ±0.022	0.437 ±0.030	0.187 ±0.014	0.611 ±0.008	0.534
	1	□	■	0.651 ±0.032	0.559 ±0.029	0.301 ±0.007	0.684 ±0.003	
	2	□	■	0.673 ±0.029	0.582 ±0.017	0.310 ±0.014	0.701 ±0.008	
	3	□	■	0.659 ±0.019	0.586 ±0.026	**0.324** ±0.019	**0.708** ±0.012	
1	0	□	□	0.547 ±0.019	0.402 ±0.033	0.156 ±0.029	0.549 ±0.015	0.516
	1	□	□	0.651 ±0.017	0.552 ±0.012	0.288 ±0.008	0.680 ±0.007	
	2	□	□	0.664 ±0.031	0.566 ±0.012	0.288 ±0.017	0.693 ±0.011	
	3	□	□	0.663 ±0.008	0.561 ±0.014	0.291 ±0.012	0.703 ±0.007	
$\frac{1}{\sqrt{d}}$	0	□	□	0.566 ±0.014	0.431 ±0.015	0.182 ±0.023	0.606 ±0.012	0.534
	1	□	□	0.655 ±0.015	0.567 ±0.010	0.286 ±0.018	0.678 ±0.013	
	2	□	□	0.678 ±0.014	0.591 ±0.026	0.307 ±0.007	0.702 ±0.007	
	3	□	□	0.684 ±0.022	0.595 ±0.017	0.321 ±0.016	0.698 ±0.015	

B.6 GAT layers

In Figure 7, we show the effect of adding dropout after each GAT layer. With $a = 1$ and $k = 1$, dropout seemingly makes the performance trend more stable at $L_{\text{GNN}} = 3$. However, the top performance is still lower in chart [0,0] compared to chart [0,2] of the table. When looking at $L_{\text{GNN}} = 2$ in chart [1,2], the training triggers early stopping with very low final performance, compared to [1,0]. In general, then, dropout seems to have a negative effect when applied to the GAT layers. However, normalizing the scores by setting $a = 1/\sqrt{d}$ attenuates the instability and drop in performance it causes.

[6]https://docs.pytorch.org/docs/stable/nn.init.html#torch.nn.init.xavier_normal_

Table 6: Performance in terms of F_1-measure when ablating over different output dimension for the parser's MLPs ($\phi = \checkmark$, $\mathbf{e}_i^{tag} = \checkmark$). Best in bold.

a	d_{MLP}	ADE	CoNLL04	SciERC	ERFGC
(L_ψ, h_ψ) =		(3, 200)	(3, 400)	(3, 300)	(3, 400)
1	100	0.662 ±0.027	0.579 ±0.030	0.302 ±0.018	0.709 ±0.008
	300	0.658 ±0.018	0.562 ±0.021	0.299 ±0.023	0.705 ±0.011
	500	0.657 ±0.018	0.566 ±0.019	0.281 ±0.016	0.701 ±0.009
$\frac{1}{\sqrt{d}}$	100	0.686 ±0.025	0.586 ±0.013	0.318 ±0.017	**0.712** ±0.008
	300	0.680 ±0.018	**0.602** ±0.017	**0.320** ±0.013	0.708 ±0.008
	500	**0.685** ±0.019	0.600 ±0.015	0.311 ±0.009	0.707 ±0.004

Table 7: Performance in terms of F_1-measure when ablating over different hidden sizes for the parser's stacked BiLSTMs ($\phi = \checkmark$, $\mathbf{e}_i^{tag} = \checkmark$). Best in bold.

a	h_ψ	ADE	CoNLL04	SciERC	ERFGC
(L_ψ, d_{MLP}) =		(3, 100)	(3, 300)	(3, 300)	(3, 300)
1	100	0.682 ±0.021	0.580 ±0.031	0.291 ±0.021	0.693 ±0.011
	200	0.662 ±0.027	0.582 ±0.006	0.286 ±0.032	0.703 ±0.007
	300	0.674 ±0.029	0.585 ±0.026	0.299 ±0.023	0.703 ±0.006
	400	0.663 ±0.032	0.562 ±0.021	0.289 ±0.038	0.705 ±0.011
$\frac{1}{\sqrt{d}}$	100	0.685 ±0.020	0.600 ±0.018	0.302 ±0.013	0.698 ±0.008
	200	**0.686** ±0.025	**0.610** ±0.018	0.314 ±0.019	0.702 ±0.008
	300	0.678 ±0.012	0.599 ±0.012	**0.320** ±0.013	**0.711** ±0.005
	400	0.674 ±0.011	0.602 ±0.017	0.306 ±0.016	0.708 ±0.008

B.7 Transformer encoder layers

In this section we show the effect of using Transformer encoder blocks rather than BiLSTM layers. In this setting, we use a learning rate of $\eta = 10^{-4}$, similarly to the full fine-tuning setting, since higher learning rates are too high for Transformer encoders. Accordingly, we also raise the number of training steps to 10,000, compared to the 2,000 used in the main setting.

Table 10 shows the results for $L \in \{1, 2, 3\}$, which in general are lower than the ones obtained with a single BiLSTM layer (see Table 3. With one BiLSTM layer the model has $|\Theta| \approx 7$M parameters. When using encoder layers with the same size as those of BERT_{base}, the model has a higher number of parameters, with $|\Theta_{L=1}| = 11$ M, $|\Theta_{L=2}| = 18$ M, and $|\Theta_{L=3}| = 25$ M. Moreover, this performance is obtained after training for five times as many steps. This supports the use of BiLSTMs in the literature for this task and demonstrates the efficacy and efficiency of using BiLSTM layers compared to Transformer layers.

It is also interesting to notice that in this case the effect of normalization does not seem to produce any significant performance boosts. This behavior is very similar to the one observed when fully fine-tuning the models in the following Section B.8. One possible explanation for this is that the (slightly) higher parameter count diminishes the effects of normalization even with a single Transformer encoder layer. The normalization layers found in each of the

Table 8: Ablation over the best hyperparameters (h_ψ, d_{MLP}) and $\phi = \checkmark$ and $\mathbf{e}_i^{tag} = \checkmark$, using LayerNorm layers $\text{LN}_\psi \in \{\checkmark, \times\}$ and Xavier uniform/normal initialization $I_{par} \in \{\mathcal{U}, \mathcal{N}\}$ for the parser. The first quarter of the table is equivalent to Table 3. Best in bold, second-best underlined.

I_{par}	LN_ψ	a	L_ψ	ADE	CoNLL04	SciERC	ERFGC	Mean
$(h_\psi, d_{\text{MLP}}) =$				(200, 100)	(400, 300)	(300, 300)	(400, 300)	
$\mathcal{U}$	□	1	0	0.541 ±0.021	0.399 ±0.024	0.147 ±0.049	0.548 ±0.010	0.515
	□		3	0.662 ±0.027	0.562 ±0.021	0.299 ±0.023	0.705 ±0.011	
	□		1	0.657 ±0.011	0.556 ±0.021	0.282 ±0.009	0.676 ±0.010	
	□		2	0.667 ±0.011	0.573 ±0.025	0.273 ±0.010	0.694 ±0.010	
	□	$\frac{1}{\sqrt{d}}$	0	0.567 ±0.014	0.438 ±0.033	0.181 ±0.027	0.612 ±0.008	**0.541**
	□		1	0.668 ±0.017	0.597 ±0.015	0.299 ±0.019	0.692 ±0.009	
	□		2	0.676 ±0.019	0.596 ±0.014	0.312 ±0.011	0.699 ±0.009	
	□		3	0.686 ±0.025	0.602 ±0.017	**0.320** ±0.013	0.708 ±0.008	
	■	1	0	0.545 ±0.018	0.399 ±0.024	0.151 ±0.014	0.548 ±0.010	0.468
	■		1	0.680 ±0.014	0.554 ±0.042	0.265 ±0.028	0.686 ±0.007	
	■		2	0.679 ±0.011	0.578 ±0.033	0.260 ±0.020	**0.715** ±0.012	
	■		3	0.680 ±0.013	0.117 ±0.261	0.060 ±0.133	0.569 ±0.318	
	■	$\frac{1}{\sqrt{d}}$	0	0.567 ±0.014	0.433 ±0.029	0.181 ±0.027	0.614 ±0.004	0.503
	■		1	0.664 ±0.017	0.564 ±0.037	0.282 ±0.029	0.686 ±0.004	
	■		2	**0.697** ±0.022	**0.623** ±0.019	0.300 ±0.042	0.702 ±0.011	
	■		3	0.675 ±0.019	0.239 ±0.327	0.111 ±0.152	0.703 ±0.006	
$\mathcal{N}$	□	1	0	0.545 ±0.017	0.415 ±0.014	0.155 ±0.019	0.558 ±0.009	0.520
	□		1	0.667 ±0.014	0.543 ±0.017	0.275 ±0.020	0.680 ±0.015	
	□		2	0.674 ±0.025	0.576 ±0.023	0.272 ±0.014	0.699 ±0.006	
	□		3	0.672 ±0.028	0.580 ±0.022	0.297 ±0.019	0.705 ±0.006	
	□	$\frac{1}{\sqrt{d}}$	0	0.570 ±0.013	0.454 ±0.006	0.181 ±0.023	0.613 ±0.007	<u>0.538</u>
	□		1	0.671 ±0.015	0.578 ±0.031	0.299 ±0.030	0.685 ±0.010	
	□		2	0.671 ±0.031	0.593 ±0.016	0.301 ±0.019	0.704 ±0.007	
	□		3	0.681 ±0.024	0.590 ±0.014	<u>0.315</u> ±0.009	0.702 ±0.011	
	■	1	0	0.545 ±0.017	0.415 ±0.014	0.155 ±0.019	0.558 ±0.009	0.469
	■		1	0.679 ±0.012	0.582 ±0.012	0.259 ±0.010	0.680 ±0.016	
	■		2	0.668 ±0.015	0.580 ±0.041	0.284 ±0.024	<u>0.711</u> ±0.005	
	■		3	0.679 ±0.018	0.000 ±0.000	0.000 ±0.000	<u>0.711</u> ±0.009	
	■	$\frac{1}{\sqrt{d}}$	0	0.570 ±0.013	0.454 ±0.006	0.181 ±0.023	0.613 ±0.007	0.512
	■		1	0.675 ±0.019	0.578 ±0.022	0.280 ±0.022	0.682 ±0.006	
	■		2	<u>0.687</u> ±0.021	<u>0.609</u> ±0.012	0.308 ±0.012	0.702 ±0.011	
	■		3	0.678 ±0.009	0.476 ±0.266	0.000 ±0.000	0.701 ±0.006	

Transformer layers could also be at play. However, in Section B.5 we show that interleaving BiLSTM and LayerNorm layers is not overall useful, making that this explanation less likely.

B.8 Full fine-tuning and sample efficiency

In this section, we present the results obtained when fine-tuning BERT_{base}, DeBERTa_{base}, BERT_{large}, and DeBERTa_{large} over $10k$ steps. In our previous settings, we only used a frozen BERT_{base}, whose last hidden states we fed as input to the tagger and parser. This evaluation allows us to test the effectiveness of our approach

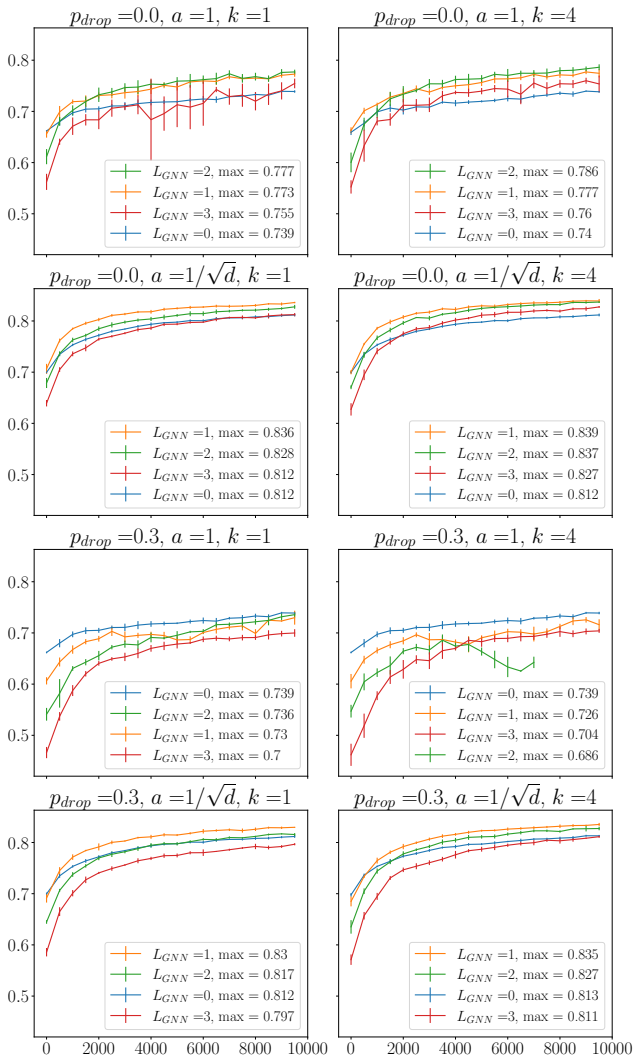


Figure 7: Test performance on SciDTB (LAS vs number of training steps), with and without dropout on the GAT layers.

Table 9: Performance for the fully fine-tuned BERT and DeBERTa models ($h_\psi = 400$, $h_{out} = 500$, $\phi = \checkmark$ and $\mathbf{e}_i^{tag} = \checkmark$).

Model	a	ADE	CoNLL04	SciERC	ERFGC	Mean
BERT_{base}	1	0.748 ±0.028	0.613 ±0.019	0.414 ±0.011	0.726 ±0.006	0.321
	$\frac{1}{\sqrt{d}}$	0.731 ±0.025	0.629 ±0.022	0.412 ±0.016	0.726 ±0.006	0.321
BERT_{large}	1	0.748 ±0.016	0.700 ±0.019	0.473 ±0.011	0.750 ±0.006	0.340
	$\frac{1}{\sqrt{d}}$	0.777 ±0.011	0.697 ±0.014	0.446 ±0.025	0.748 ±0.010	0.341
DeBERTa_{base}	1	0.754 ±0.013	0.674 ±0.014	0.429 ±0.015	0.751 ±0.002	0.332
	$\frac{1}{\sqrt{d}}$	0.761 ±0.021	0.700 ±0.013	0.425 ±0.019	0.751 ±0.010	0.338
DeBERTa_{large}	1	0.786 ±0.010	0.739 ±0.016	0.478 ±0.019	0.763 ±0.006	0.352
	$\frac{1}{\sqrt{d}}$	0.794 ±0.015	0.747 ±0.013	0.476 ±0.010	0.763 ±0.007	0.353

Table 10: Micro-F1 (SemDP) and LAS (SynDP) for the labeled edge prediction task, using $L \in \{1, 2, 3, \}$Transformer encoder layers instead of BiLSTMs. Best in bold.

L	a	ADE	CoNLL04	SciERC	ERFGC	enEWT	SciDTB
1	1	0.607	**0.492**	**0.203**	**0.608**	0.727	0.855
	$\frac{1}{\sqrt{d}}$	0.601	0.478	0.191	0.571	0.729	0.854
2	1	0.604	0.484	0.177	0.583	0.732	**0.857**
	$\frac{1}{\sqrt{d}}$	0.615	**0.492**	0.193	0.563	**0.734**	**0.857**
3	1	0.611	0.479	0.170	0.564	0.708	0.854
	$\frac{1}{\sqrt{d}}$	**0.616**	0.483	0.173	0.557	0.719	0.852

in a setting in which the number of learnable parameters is unconstrained. In this experiment, we use different learning rates for the base models ($\eta = 1 \times 10^{-4}$) and the large models ($\eta = 3 \times 10^{-5}$) and we apply gradient norm clipping with $\|\nabla\|_{max} = 1.0$. In the case of the large models, we also use a cosine schedule with warm-up over 6% of the steps. We adopt these measures because during early trials we experienced sudden mid-run gradient explosions. Note that, in this case, we do not use any downstream BiLSTMs and only vary whether we use biaffine score normalization in the parser.

Table 9 reports the performance in terms of micro-averaged F_1-measure for the best models, picked based on top validation performance, evaluated every 100 steps. As the table shows, normalizing the scores generally does not increase top performance when fully fine-tuning these models. This is in line with our theoretical results, since tuning all of their ~$100 - 400M$ parameters can compensate for the lack of biaffine score normalization.

In order to have a stronger indication of whether score normalization also works in this setting, we study the performance on the test set versus the amount of training steps. Figure 8 and Figure 9 respectively visualize the performance of the base and large models on the test set in terms of micro-averaged F_1-measure over $10k$ training steps. We still use early stopping, which is why some of the series are cut short before reaching $10k$ steps.

As Figure 8 shows, a one-tailed Wilcoxon signed-rank test finds the difference in performance throughout the training to be significantly higher when normalizing scores. Since the average test performance of the models is similar with and without normalization, the gap between the two curves being statistically significant indicates faster convergence with normalization.

For both BERT_{base} and DeBERTa_{base}, the difference in convergence speed and performance is statistically significant on ADE, CoNLL04, and SciERC. The same is true for BERT_{large} only on ADE. However, the effect is statistically significant with DeBERTa_{large} for all datasets. This indicates our score normalization approach can produce positive effects in terms of sample efficiency also when fine-tuning models with hundreds of millions of parameters.

B.9 LLM performance comparison

Table 11 reports the results for the parsing experiment, carried out using *Mistral-7B-Instruct-v0.3*, an instruction-tuned LLM with 7B parameters. We train the model using LoRA [13] and only target the query, key, and value matrices of the decoder blocks. With LoRA, model weight updates are carried out as:

$$W_0 \leftarrow W_0 + \frac{\alpha}{r} BA$$

where $B \in \mathbb{R}^{d_1 \times r}$ and $A \in \mathbb{R}^{r \times d_2}$, $r \ll d_1, d_2$. We set $\alpha = r = 16$, where the α modifies the updates to the LoRA similarly to a learning rate, while r is the rank of the low-rank projection. With these hyperparameters, the total amount of learnable parameters is $9.4M$, an amount comparable to those of the main setting ($7–14M$ BiLSTM parameters).

We train the model for 200 steps with a batch size of 8. We use this number of steps because it results in similar training times, compared to the main setting models, which have an upper bound of ~7 minutes. We use a learning rate of $\eta = 2 \times 10^{-4}$ with 5 warm-up steps, weight decay of 0.01, and AdamW [20] as the optimizer.

During training we provide the model with one ICL example chosen at random from the training split of the used dataset, followed by the training sample from which to extract entities and edges in RDF triple format. The ICL and target sample are never the same. We only use one example in the prompt because ERFGC, enEWT and SciDTB comprise very long samples, which means multiple samples do not fit even when using a context size 8,192 tokens, with bigger windows not fitting in the available VRAM (96 GB on a single NVIDIA H100).

Note that UAS and LAS require edge predictions for all tokens in the sentence, and not just for the relevant entity triples. Since producing all predictions using an LLM is unfeasible both compute- and performance-wise, in this case we use F_1-measure also for enEWT and SciDTB.

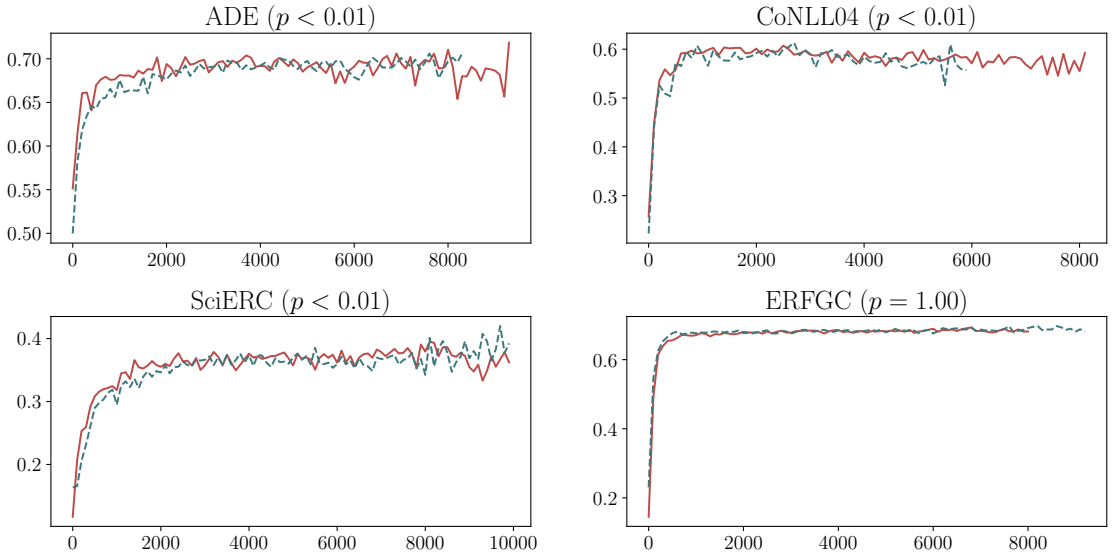
On ADE, the LLM's performance is higher than the graph-based parsers of our base setting, but still lower than the results obtained when fully fine-tuning DeBERTa_{base}, BERT_{large}, and DeBERTa_{large}, as shown in Section B.8. In the case of CoNLL04 and SciERC, the performance is very similar to our base setting. Conversely, for more complex graphs, such as the ones comprising ERFGC, enEWT and SciDTB, the performance is considerably lower. Therefore, while LLMs of this size perform better than graph-based parser on short sentences with few entities and edges, on large graphs better performance can be obtained with ~1.7% of the total parameters.

Table 11: Micro-averaged precision, recall and F_1 for the labeled edge prediction task, using Mistral-7B-Instruct-v0.3.

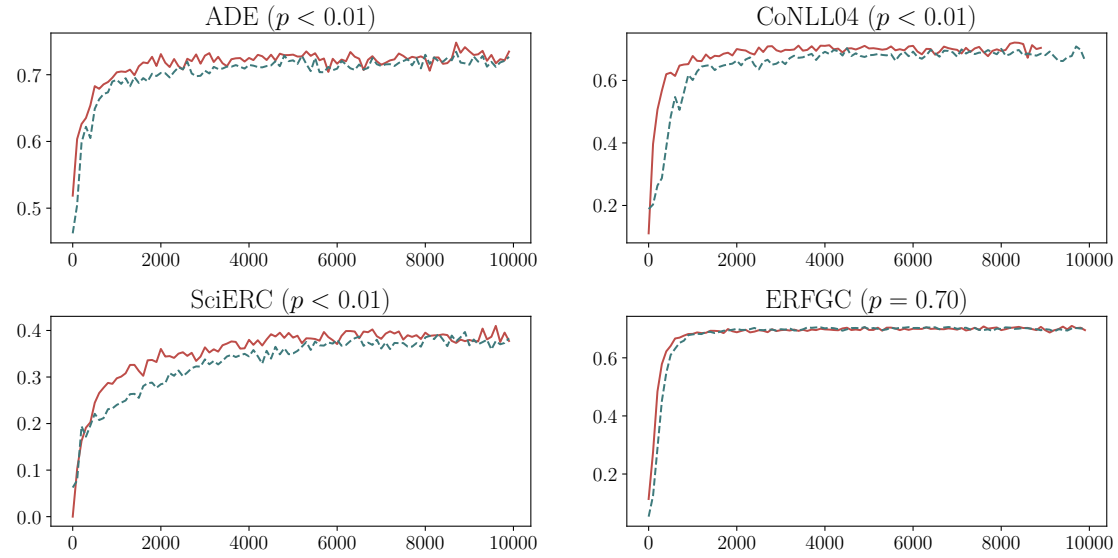
	ADE	CoNLL04	SciERC	ERFGC	enEWT	SciDTB
P	0.757	0.584	0.311	0.555	0.606	0.728
R	0.718	0.619	0.341	0.417	0.595	0.717
F_1	0.737	0.599	0.325	0.476	0.601	0.722

C Computational resources

We ran all of our experiments on a cluster of NVIDIA H100 (96GB of VRAM) and NVIDIA L40 (48GB of VRAM) GPUs, one run per single GPU. When freezing the BERT_{base} encoder and only training the BiLSTMs and the classifiers, each training and evaluation run took ~5-7 minutes, depending on the number of BiLSTM layers.



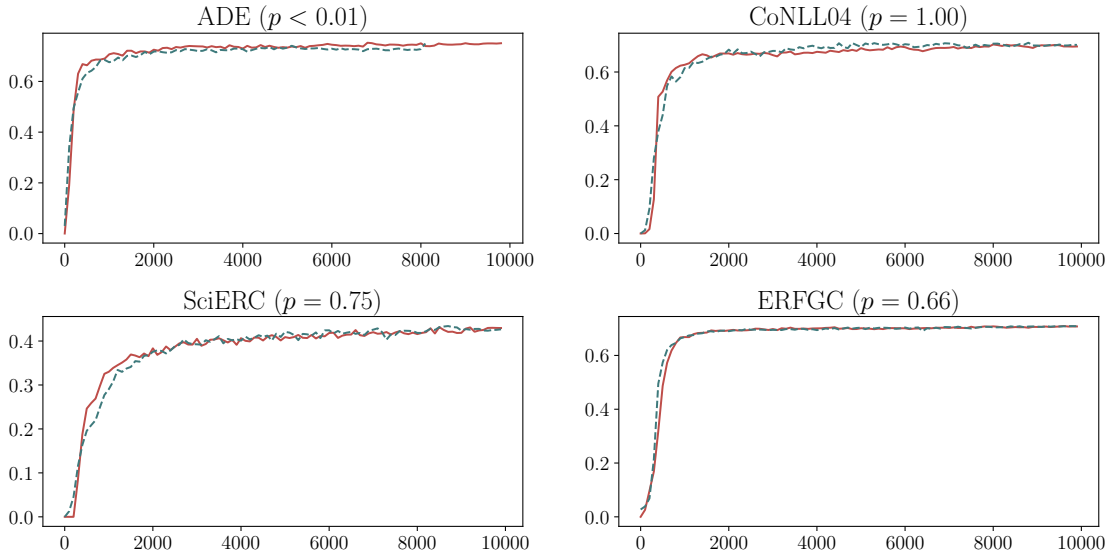
(a) BERT$_{base}$



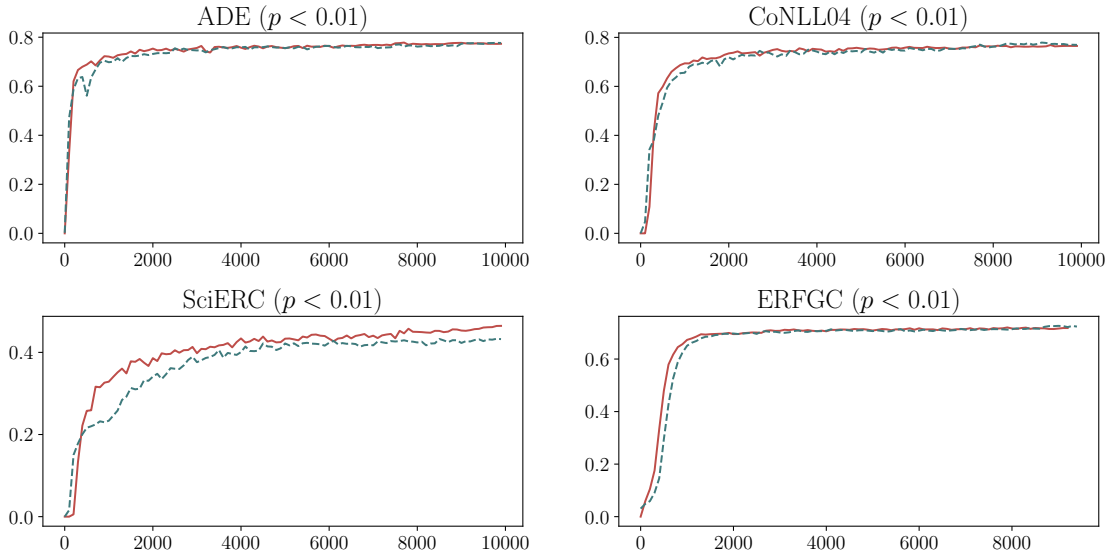
(b) DeBERTa$_{base}$

Figure 8: Performance in terms of F_1-measure vs the number of training steps for the base models on the SemDP datasets. Red = norm; blue = raw. The p-values refer to the performance being greater with score normalization (one-tailed Wilcoxon signed-rank test).

For the main setting, finding the best hyperparameters involved training and testing 6,120 models, for a total of ~600 GPU hours. When carrying out the full fine-tuning ablation (Appendix B.8), training and evaluation took ~1 hour for each of the 40 base models and ~2-3 hours for each of the 40 large models, for an additional ~140 GPU hours. In the case of Mistral-7B-Instruct-v0.3, training took between 1 and 30 minutes on an NVIDIA H100 depending on the dataset.



(a) BERT$_{large}$



(b) DeBERTa$_{large}$

Figure 9: Performance in terms of F_1-measure vs the number of training steps for the large models on the SemDP datasets. Red = norm; blue = raw. The p-values refer to the performance being greater with score normalization (one-tailed Wilcoxon signed-rank test).