Bringing Emerging Architectures to Sequence Labeling in NLP

Ana Ezquerro[1,2]
ana.ezquerro

Carlos Gómez-Rodríguez[1]
carlos.gomez

David Vilares[1]
david.vilares

[1]Universidade da Coruña, CITIC (@udc.es)
[2]Graz University of Technology, IML (@tugraz.at)

Abstract

Pretrained Transformer encoders are the dominant approach to sequence labeling. While some alternative architectures—such as xLSTMs, structured state-space models, diffusion models, and adversarial learning—have shown promise in language modeling, few have been applied to sequence labeling, and mostly on flat or simplified tasks. We study how these architectures adapt across tagging tasks that vary in structural complexity, label space, and token dependencies, with evaluation spanning multiple languages. We find that the strong performance previously observed in simpler settings does not always generalize well across languages or datasets, nor does it extend to more complex structured tasks.

1 Introduction

Sequence labeling (SL) is a problem in machine learning, and particularly in NLP, where each element in a sequence is assigned exactly one output label. A feature of SL tasks is that labels are not predicted in isolation: they often depend on neighboring inputs. To address these dependencies, various architectures have been proposed over the years to model both short- and long-range interactions between input tokens, including Conditional Random Fields (CRF; Lafferty et al., 2001), sliding-window perceptrons (Zhang and Clark, 2008), Long Short-Term Memory networks (LSTMs; Hochreiter and Schmidhuber, 1997), and Convolutional Neural Networks (CNNs; LeCun et al., 1995). Today, Transformers (Vaswani et al., 2017), pretrained on large data, are the dominant approach for tagging tasks, offering superior ability to model relations across words.

At the same time, research continues to explore both alternatives to Transformers and architectural variants of them across a range of machine learning tasks. This includes, for instance, enhanced contextualization mechanisms (xLSTM; Beck et al., 2024) and structured state-space models (Mamba; Gu et al., 2022) for language modeling, as well as diffusion models (Ho et al., 2020) and generative adversarial networks (GANs; Goodfellow et al., 2014), which were originally developed for computer vision applications. Some of these architectures have shown competitive, albeit preliminary, performance compared to Transformers in metrics such as perplexity (see again Beck et al.) and could be promising for tagging tasks. Meanwhile, others have already been adapted for sequence labeling in relatively simpler settings (e.g., Huang et al., 2023; Tong et al., 2024), typically focusing on tasks like named-entity recognition (NER), part-of-speech (PoS) tagging, or (Chinese/Japanese/Korean) word segmentation. However, evaluations have mostly remained within these flatter, coarse-grained setups, leaving open questions about how well these models generalize to more fine-grained or complex scenarios, such as recent linearizations for structured tasks involving trees (Gómez-Rodríguez and Vilares, 2018; Kitaev and Klein, 2020; Amini et al., 2023) or graphs (Ezquerro et al., 2024b).

Contribution We investigate alternative architectures beyond pretrained Transformers for sequence labeling, focusing on architectures not originally designed for token-level classification but with potential for this task. Specifically, we explore how these models can be adapted to capture linguistic structure in sequence-labeling problems, considering tasks of varying complexity, output label spaces, and dependency spans. Rather than aiming to outperform Transformers universally, the goal is to better understand the relative strengths and limitations of these architectures across different problem types. Our results show that bidirectional xLSTM architectures are generally superior to the traditional BiLSTMs across tasks and datasets, although they still fall behind Transformers. Diffusion tagging and state-space mod-

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 4886–4909
March 24-29, 2026 ©2026 Association for Computational Linguistics

els trail the baseline, suggesting they may be less suited for NLP tagging. Finally, adversarial tagging yields a noteworthy result, consistently rivaling or surpassing the Transformer baseline across diverse tasks, including the most complex structured settings. Our code is publicly available at https://github.com/anaezquerro/separ.

2 Background

We now outline concepts in tagging and modeling advances.

2.1 Sequence labeling for NLP

Classical problems like PoS tagging or lemmatization naturally align with the definition of sequence labeling. Others, such as NER, chunking (Ramshaw and Marcus, 1995), segmentation (Hacioglu et al., 2004), semantic role labeling (Strubell et al., 2018) and slot filling (Li et al., 2020), can also be framed as tagging problems, typically using lightweight encoding schemes that assign token-level labels (e.g. IOB encoding). Despite gains from pretrained Transformers, many tasks, especially simpler ones, were already tractable with shallow, easier-to-deploy models.

Similarly, previous efforts focused on reformulating tree- and graph-structured tasks as tagging through linearizations. However, pre-neural models struggled with these problems. Spoustová and Spousta (2010) showed that linguistically informed linearizations trained with pre-neural models performed impractically compared to the state of the art at the time. This limitation of earlier models started to change with context-aware encoders based on BiLSTMs or Transformers, which have recently revived sequence labeling for structured prediction, though effectively modeling such structure was not immediate (Li et al., 2018). In addition, they open up new possibilities for evaluating alternative sequence tagging architectures under more demanding testbeds, with large output spaces and long-range dependencies. In this context, linearization strategies have been proposed for both continuous (Gómez-Rodríguez and Vilares, 2018; Kitaev and Klein, 2020; Amini and Cotterell, 2022) and discontinuous constituent parsing (Vilares and Gómez-Rodríguez, 2020), as well as for projective and non-projective syntactic dependency parsing (Strzyz et al., 2019; Amini et al., 2023) and, recently, graph parsing (Ezquerro et al., 2024b).

2.2 Sequence modeling

While LSTMs and Transformers are the standard encoders for token-level classification tasks, recent years have seen growing interest in alternative techniques—some of which still leverage self-attention—and architectures that replace the Transformer with different contextualization systems. Although many of these methods were not initially designed for sequence labeling, we now examine their potential.

GANs, created for image generation, have been extended to text generation in NLP, addressing the challenge posed by the discrete nature of language (Kusner and Hernández-Lobato, 2016; Yu et al., 2017). However, fewer studies have explored their use in non-generative tasks, which require generating or refining structured outputs rather than free-form text. Notably, Parnow et al. (2021) trained a GAN-based tagging system to enhance grammatical error correction by enriching the learning process with generated errors. Tong et al. (2024) improved word segmentation and NER with a GAN-based framework, using the generator as a labeler and a discriminator to guide accurate sequences.

Similarly, diffusion models (Ho et al., 2020) are often used for generative NLP tasks (He et al., 2023; Han et al., 2023). While several studies improve denoising embeddings in latent space (Gao et al., 2024; Zhou et al., 2024) to enhance text-to-text generation (Shi et al., 2023; Liu et al., 2024), few have explored their potential for tagging. Notably, some recent work has adapted diffusion processes for NER (Shen et al., 2023) and PoS tagging (Huang et al., 2023).

In addition, recent work has explored alternative architectures replacing self-attention. Gu et al. (2022) introduced structured-state space models (SSM) for linear-time language modeling, addressing the quadratic complexity of Transformers. Beck et al. (2024) introduced the xLSTM, a variant of the LSTM with parallelizable capabilities and better modeling of dependencies. Still, xLSTM has been mainly evaluated on language modeling and bioinformatics (Heidari et al., 2025; Sun et al., 2025), not on tagging for NLP.

3 Sequence labeling architectures

Current SL models typically consist of two main components. First, an encoder $\mathcal{E}_\theta : \mathcal{V}^n \to \mathbb{R}^{n\times d}$, usually a pretrained masked language model, contextualizes an input sentence $W = (w_1 \cdots w_n) \in$

$\mathcal{V}^n$ in a d-dimensional latent space. Then, each token embedding is passed through a decoder[1] $\mathcal{D}_\phi : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times |\mathcal{L}|}$ to produce a probability distribution over the label set $\mathcal{L}$.

This work examines alternative formulations of tagging along two complementary directions: modifying the learning strategy (typically a supervised input-to-label mapping) and the main underlying architecture for encoding (typically a pretrained language model). We begin by discussing strategies for modeling the label space $\mathcal{L}$, focusing on stable diffusion (§3.1) and adversarial learning (§3.2). Although these techniques have been applied to tagging (Huang et al., 2023; Tong et al., 2024), prior evaluations were often limited in scope, typically restricted to simpler tasks. We then explore SSM models and xLSTMs as alternatives to Transformers for sequence contextualization in tagging problems (§3.3).

3.1 Diffusion Tagging

We first present a sequence labeler using diffusion tagging, based on a bit-tag converter (Huang et al., 2023) to handle discrete sequential outputs. Huang et al. (2023) used the denoising diffusion implicit model (DDIM) sampling[2] to directly predict the target data, thus deviating from the original step-by-step denoising process of diffusion models. To better align the principles of stable diffusion to neural tagging, we adopt the bit-tag converter of Huang et al. (2023) but propose a conditional diffusion model that iteratively denoises a random signal by learning the added noise during the forward process, closely following the original denoising process for a fuller evaluation of diffusion in tagging.

Bit conversion The Bit-Tag converter (BT) by Huang et al. (2023) transforms an input tag sequence $(\ell_1 \cdots \ell_n) \in \mathcal{L}^n$ into a sequence of bits to treat the output as a continuous signal. Formally, the forward transformation (tag2bit) maps each integer identifier of a discrete set $\mathcal{L}$ into a sequence of $m = \lceil \log_2 |\mathcal{L}| \rceil$ bits. For instance, given $\mathcal{L}_4 = \{0, 1, 2, 3\}$, the tag2bit operation transforms each label into a sequence of 2 bits, so tag2bit$(\mathcal{L}_4) = \{00, 01, 10, 11\}$. The reverse process (bit2tag) transforms a sequence of m bits into an integer in the range $[0, 2^m - 1]$.[3]

Forward process Diffusion models gradually add Gaussian noise to a clean sample $\mathbf{x}_0$ during T timesteps, and train a neural network to model the reverse process, progressively denoising the sample. When adapting diffusion models to discriminative tasks the input to the forward process is the actual target, and the input sentence is fed as a conditional signal. In this case, $\mathbf{x}_0$ is the noise-free bit representation of the target sequence of labels. Then, from a noise schedule $\beta_1 \cdots \beta_T$, where $\beta_i < \beta_{i+1}$ and $\beta_i \in (0, 1)$, $\forall i = 1 \cdots T$, the sequence of latent variables $\mathbf{x}_1 \cdots \mathbf{x}_T$ follows a Markov process, such that each latent variable is generated by adding Gaussian noise to the previous one (Equation 1). When defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_t$, the forward process is defined conditioned on $\mathbf{x}_0$ (Equation 2).

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) \sim \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \quad (1)$$

$$q(\mathbf{x}_t|\mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \quad (2)$$

The diffusion tagger (DiT) trains a neural network to estimate the noise component present in the latent variable at each timestep, using: (i) the latent variable $\mathbf{x}_t$, (ii) the timestep t and (iii) the input tokens $(w_1 \cdots w_n)$ as a conditional signal. During training, timesteps are uniformly sampled to generate latent variables following Equation 2. Following Ho et al. (2020), our model is tasked to minimize the MSE loss between the real and predicted noise.

Algorithm 1 and Figure 4a (§A) show the forward process for a sample (w, ℓ). The token encoder $\mathcal{E}_\theta$ is an encoder-only network that contextualizes tokens in the input sequence. The decoder $\mathcal{D}_\phi$ is a Transformer-based architecture that accepts as input the (noised) sample, the timestep t and the contextualized embeddings. Following the bit conversion described above, each tag is represented as an m-dimensional binary vector. At each training step, a timestep t is sampled uniformly to compute the latent variable $\mathbf{x}_t$. The decoder is then trained to estimate the noise added to $\mathbf{x}_t$.

Denoising process For our diffusion tagger we adopt DDIM sampling (Song et al., 2021), which allows skipping timesteps to increase inference speed. Since each latent variable is defined in the forward

[1] E.g., either a feed-forward network (FFN) with a non-linear activation or more complex projection heads (e.g., CRFs) could be used.

[2] Song et al. (2021) proposed a faster, more stable reformulation of the original denoising process (Ho et al., 2020, DDPM) using non-Markovian deterministic sampling.

[3] If $\mathcal{L}$ is not power of 2, bit2tag may yield integers outside $\mathcal{L}$. These are removed and replaced by the most common tag.

Algorithm 1: Forward process.

```
Noise schedule $\{\bar{\alpha}_1 \cdots \bar{\alpha}_T\}$;
Word encoder $\mathcal{E}_\theta : \mathcal{V}^n \to \mathbb{R}^{n \times d}$;
Decoder $\mathcal{D}_\phi : (\mathbb{R}^{n \times m}, \mathbb{N}, \mathbb{R}^{n \times d}) \to \mathbb{R}^{n \times m}$;
foreach $(w, \ell)$ in dataset do
    $\mathbf{w} = \mathcal{E}_\theta(w)$ ;            /* word embeddings */
    Initial signal: $\mathbf{x}_0 = \texttt{tag2bit}^*(\ell)$;
    $t \sim \mathrm{Unif}(1, T)$ ;        /* sample timestep */
    $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ;            /* Gaussian noise */
    Latent variable: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{e}$;
    Gradient descent step on:
      $\nabla_{\theta,\phi} = \|\mathbf{e} - \mathcal{D}_\phi(\mathbf{x}_t, t, \mathbf{w})\|^2$
end
```

Algorithm 2: Denoising process.

```
Input: Sample $W = (w_1 \cdots w_n) \in \mathcal{V}^n$ and number
       of skipped inference steps $s$.
Output: Estimated tag sequence $\tilde{\ell}$.
$\mathbf{w} = \mathcal{E}_\theta(w)$ ;                  /* word embeddings */
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ;                  /* Gaussian noise */
$t \leftarrow T$;
while $t > 0$ do
    $\tilde{\mathbf{e}}_t = \mathcal{D}_\phi(\mathbf{x}_t, t, \mathbf{w})$;
    $k = t - s$ if $t - s > 0$ otherwise 0;
    $\tilde{\mathbf{x}}_k = \frac{\sqrt{\bar{\alpha}_k}}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\tilde{\mathbf{e}}_t) + \sqrt{1 - \bar{\alpha}_k}\tilde{\mathbf{e}}_t$;
    $t \leftarrow k$
end
$\tilde{\ell} = \texttt{bit2tag}^*(\tilde{\mathbf{x}}_0)$ ;        /* bit conversion */
return $\tilde{\ell}$
```

process as $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\mathbf{e}$, the estimation of a previous latent $\mathbf{x}_k$, where $k < t$, can be obtained with the estimated noise from $\mathcal{D}_\phi$. Algorithm 2 and Figure 4b (§A) show the denoising process using an hyperparameter s, controlling how many timesteps are skipped in the reverse process.

Model architecture We use a pretrained masked language model for the encoder module $\mathcal{E}_\theta$, recovering the last hidden states as token embeddings. The decoder $\mathcal{D}_\phi$ learns the added noise from a latent variable $\mathbf{x}_t$, the token embeddings and the timestep t, which represents the noise level applied to $\mathbf{x}_t$. We adopt the DiT block proposed by Huang et al. (2023), which relies on a learnable embedding layer τ to represent the timestep t. The latent variable $\mathbf{x}_t$, the word embeddings $\mathbf{w}$ and the time embedding $\tau(t)$ are merged and fed to a stack of Transformer layers with residual connections. The final layer has an output dimension of m with a linear activation to predict the added noise.

3.2 Adversarial Tagging

Next, we follow the approach by Tong et al. (2024) to build an adversarial tagger composed of two modules: a generator G_ψ and a discriminator D_φ. In their concept of adversarial training for tagging, the generator receives a sentence and is trained to generate the tag sequence, while the discriminator evaluates the generator's predictions against the ground-truth tags to identify incorrect outputs. To simplify the original setup and enable clearer comparison with other taggers, we remove the CRF module and reduce the architecture of the discriminator to a 2-layered BiLSTM stack.

Generator The generator $G_\psi : \mathcal{V}^n \to \mathbb{R}^{n \times |\mathcal{L}|}$ is an encoder-decoder neural architecture that learns the real tags from an input sentence, as traditional SL approaches. The encoder $G_\psi^{\mathcal{E}} : \mathcal{V}^n \to \mathbb{R}^{n \times d}$ maps each token into a learned latent space, and the decoder $G_\psi^{\mathcal{D}} : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{L}|}$ independently projects each embedding into the learned tag distribution. The generator loss is defined as the cross-entropy between the real tags and the predicted distribution.

Discriminator The discriminator $D_\varphi : (\mathcal{V}^n, \mathbb{R}^{n \times |\mathcal{L}|}) \to \mathbb{R}^n$ takes as input the sequence of words and an estimated distribution over $\mathcal{L}$; and outputs a similarity score measuring how close the predicted distribution is to the true distribution $p(\ell|w)$. Let $G_\psi(w) = (\tilde{\boldsymbol{\ell}}_1, ..., \tilde{\boldsymbol{\ell}}_n)$ be the predicted distribution of the generator and $\mathbf{L} = (\boldsymbol{\ell}_1, ..., \boldsymbol{\ell}_n) = \texttt{onehot}^*(\ell_1, ..., \ell_n)$ the one-hot representation of the real tag sequence. To ease backpropagation, we apply the Gumbel-Softmax relaxation (Jang et al., 2017) to smooth the one-hot representation of the target tags, following Tong et al. (2024). The discriminator loss models valid tag sequences conditioned on the input to spot incorrect tags. Intuitively, it approximates $D_\varphi(w, \mathbf{L})$ to $\mathbf{1}$, and $D_\varphi(w, G_\psi(w))$ to $\mathbf{s} = (s_1, .., s_n)$, where each value is defined as in Equation 3:

$$s_i = \begin{cases} 1 & \text{if } \arg\max\{\tilde{\boldsymbol{\ell}}_i\} = \ell_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

using $\mathcal{H}$ as the cross-entropy loss (Equation 4):

$$\begin{aligned} L_{D,p} &= \mathcal{H}\big(D_\varphi(w, \mathbf{L}), \mathbf{1}\big) \\ L_{D,G} &= \mathcal{H}\big(D_\varphi(w, G_\psi(w)), \mathbf{s}\big) \end{aligned} \tag{4}$$

Adversarial training To replicate the adversarial dynamics of GANs, we compute the adversarial loss (Equation 5) to guide G_ψ to generate distributions that challenge D_φ. The generator loss L_G (Equation 6) includes an hyperparameter λ that controls the influence of the adversarial component during training. By adjusting λ, the generator is encouraged not only to match the gold tags but also to produce outputs that fool the discriminator. Figure 5 (§A) visualizes our adversarial training.

$$L_A = \mathcal{H}(D_\varphi(w, G_\psi(w)), \mathbf{1}) \tag{5}$$

$$\begin{aligned} L_G &= \mathcal{H}(G_\psi(w), \ell) + \lambda L_A \\ L_D &= L_{D,p} + L_{D,G} \end{aligned} \tag{6}$$

Model architecture For the generator module, we rely on a encoder-decoder architecture, where the decoder is a FFN with a non-linear activation. For the discriminator, we use a lightweight BiLSTM-based encoder[4] to contextualize the sequence of token embeddings and tag logits, followed by a FFN to validate the correctness of the input tag distribution.

3.3 Alternatives for sequence modeling

We now describe encoder architectures beyond Transformers that, to our knowledge, remain untested for tagging tasks.

xLSTM Beck et al. (2024) recently proposed a new recurrent unit inspired on the LSTM unit that deals with the main drawbacks of its ancestor when modeling long dependencies and enabling parallelization. The xLSTM relies on two blocks: the sLSTM, which deals with the first problem by stabilizing the information flow in the forget gate; and the mLSTM, which enables GPU parallelization by replacing the vectorial form of the hidden states with learnable matrices (resembling the self-attention operation). By stacking xLSTM blocks, we explore xLSTM-based encoders for contextualizing input sentences for tagging tasks. Additionally, inspired on the BiLSTM design (Graves and Schmidhuber, 2005), we explore the **BixLSTM**, which processes with two different xLSTM units an input sequence from left-to-right and right-to-left and concatenates their representations.

MAMBA-2 (SSD) Building on the structured state-space (S3) framework, Gu et al. (2022) introduce the structured state-space sequence model (S4), which captures long-range dependencies with linear time and space complexity by parameterizing the dynamics using diagonal state matrices and computing convolution kernels in the frequency domain, thus enabling efficiency and expressiveness to sequence modeling. More recently, Gu and Dao (2023) proposed MAMBA, a selective state-space model (S6) extended from S4 with dynamic input-dependent weights and gating mechanisms. As part of our comparison, we adopt MAMBA-2 (Dao and Gu, 2024), an improved version of S6 with architectural refinements that address the numerical instability and throughput limitations of earlier SSMs, in the encoder architecture to examine how its state-space dual framework (SSD) behaves in relation to Transformer-based models.

4 Experiments

We evaluate these architectures on a multilingual benchmark covering multiple tasks, opting for datasets that cover scenarios of varying complexity, different output vocabulary sizes and a range of token dependencies, as these factors may influence how the underlying architecture affects performance. See §B (Table 6) for details on the datasets used, and §C for examples of the parsing linearizations introduced in next paragraphs. We have aimed to maintain a relatively homogeneous set of languages across tasks, but this was not always possible due to dataset availability or evaluation setup.

PoS tagging We use datasets with coarse- and fine-grained tag sets, as well as morphologically rich tags for languages with complex morphology: eight Universal Dependencies (UD) treebanks (Nivre et al., 2020), the Penn Treebank (PTB; Marcus et al., 1993), the Chinese Treebank (CTB; Xue et al., 2005), and the Statistical Parsing of Morphologically Rich Languages datasets (SPMRL; Seddah et al., 2014).[5]

NER We evaluate the following datasets spanning diverse languages using the BIO scheme annotation: CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003), BioNLP (Pyysalo et al., 2013), DrugsNER (HuggingFace, 2024), EIEC (Alegria et al., 2004), GermEval-2014 (Benikova et al., 2014), HiNER (Murthy et al., 2022), JBNLPA (Collier and Kim, 2004), KLUE (Park et al., 2021), NYTK-NerKor (Simon and Vadász, 2021), Webbnyheter (Språkbanken Text, 2024), Weibo-NER (Peng and Dredze, 2015) and WikiNER (Nothman

[4] Initial experiments showed no significant performance drop when reducing discriminator size, matching results without doubling the network as in Tong et al. (2024).

[5] For SPMRL, we predict PoS tags as a single task to maintain a consistent setup across all experiments. While these tags are typically decomposable into independent morphological components suitable for multi-task learning, our simplified approach explains why the reported performance is lower than standard benchmarks.

et al., 2012).

Constituent parsing We use the PTB (Marcus et al., 1993), the CTB (Xue et al., 2005), and the SPMRL datasets (Seddah et al., 2014). To cast constituent parsing as tagging, we adopt two linearizations: the relative encoding (**R**, Gómez-Rodríguez and Vilares, 2018), which captures the difference in common ancestors between words, and the tetratagging (**T**, Kitaev and Klein, 2020), based on child direction in a binary tree. These represent the two families of constituent linearizations: depth- and transition-based.

Dependency parsing We use the same eight UD treebanks (Nivre et al., 2020) as for PoS tagging. We also evaluate the dependency taggers on the PTB and CTB using the dependency conversion proposed by Marneffe and Manning (2008). Among existing linearizations to cast dependency parsing as a tagging task, the absolute encoding (**A**, Strzyz et al., 2019) is the simplest, as each label independently encodes the head of a word. Bracketing encodings represent trees using balanced bracket strings distributed across labels. In the case of naive bracketing (**B**, Strzyz et al., 2019), the label set is unbounded; we also consider the bounded 4-bit and 7-bit variants ($\mathbf{B}_4$, $\mathbf{B}_7$), which reduce the label space (Gómez-Rodríguez et al., 2023). Finally, hexatagging (**H**, Amini et al., 2023) encodes a constituency-based transformation of the dependency tree.

Graph parsing While linearizations for dependency trees are well studied, Ezquerro et al. (2024b) recently proposed unbounded and bounded variants for dependency graphs, a more expressive formalism allowing reentrancies, cycles, and non-connectivity. With k being the number of planes (see §C.3), we use the relative (**R**), bracketing with $k = 3$ (**B**), $4k$-bit with $k = 4$ ($\mathbf{4}\boldsymbol{k}$) and $6k$-bit ($\mathbf{6}\boldsymbol{k}$) encodings and conducted experiments in multiple languages from the SDP (Oepen et al., 2015) and IWPT (Bouma et al., 2021) corpus.

Evaluation We use accuracy for PoS tagging; mention-level F1 score for NER; UAS and LAS for dependency parsing; labeled F1 with the `EVALB`[6] tool and the `COLLINS.prm` and `SPMRL.prm` parameter file for constituency parsing; and labeled F1 with the `sdp-toolkit`[7] for graph parsing.

Model configuration Our diffusion and adversarial taggers finetune XLM-RoBERTa$^{\mathrm{L}}$ (XLM; Liu et al., 2020) to produce word embeddings. The diffusion tagger uses XLM as the encoder and stacks 6 DiT blocks with 16 heads as the decoder, where the bit sequence is introduced as target and the word and time embeddings as conditional signal. We set $T = 100$ and $s = 10$ steps for inference and a linear variance schedule with $\beta_1 = 0.002$ and $\beta_T = 0.03$, in order to adjust the noise level to the bit range in larger timesteps. The adversarial tagger finetunes a XLM encoder for the generator, stacked with a FFN to output the tag distribution, and a 2-layered BiLSTM stack for the discriminator, stacked with a FFN to predict the correctness of the two input sequences. The hyperparameter λ is fixed to 1. For the xLSTM encoder, we stacked four xLSTM blocks of hidden size $d = 400$, where each block consists of an mLSTM followed by an sLSTM. To allow bidirectionality, the BixLSTM block processes the input sequence in both directions using two xLSTMs (each d=200), and outputs the concatenation of their representations to the next layer. For the SSD-based encoder, we finetuned MAMBA2-370M (Dao and Gu, 2024) using a FFN to predict the final tag sequence.

Baseline We adopt a standard tagging architecture as our baseline, consisting of an XLM encoder followed by a FFN to predict output labels. For tasks naturally formulated as sequence labeling, this architecture serves as the primary benchmark. However, for parsing tasks, we additionally include established paradigms; specifically, we compare against the span-based model proposed by Kitaev et al. (2019) and the biaffine parser (Dozat and Manning, 2017).

5 Analysis of results

Tables 1 and 2 show PoS and NER accuracies, with relatively small but still unbalanced label sets and weak hierarchies. Tables 3, 4 and 5 cover constituent, dependency and graph parsing, respectively.

Table 1 breaks down the PoS tagging results using the universal, language-specific (if available), and rich PoS tags of the UD and SPMRL treebanks. The first two columns summarize the performance of the (x)LSTM-based (non-pretrained) models: although the xLSTM outperforms the LSTM in only 6 out of 22 experiments, the BixLSTM surpasses the BiLSTM in 19 out of 22 PoS tagging

[6] nlp.cs.nyu.edu/evalb.

[7] github.com/semantic-dependency-parsing/toolkit.

tasks. Overall, the BixLSTM is the best non-pretrained encoder in 15 out of 22 experiments, showing an improvement over its predecessor for simple tagging tasks, although it still falls short of the baseline. Among the pretrained models (DiT, GaT, SSD), the adversarial tagger achieves the best scores in 20 out of 22 experiments (DiT is selected only for the universal German tag, and xLSTM for the rich Hungarian tag), and is the only model to offer competitive performance against the baseline architecture as the top tagger in 9 out of 22 experiments, consistently trailing on others by less than 1 accuracy point.

		LSTM		**DiT**	**GaT**	**SSD**	**Base**
PTB		95.97	96.92ϟ	95.83	**97.91**	97.33	97.97
CTB		92.71	93.55ϟ	97.10	**97.96**	93.38	97.79
de	U	90.31	91.52	**96.62**	96.21	93.16	96.79
	X	89.99	92.26ϟ	94.62	**97.42**	92.67	97.53
	R	68.84ϟ	71.48ϟ	65.22	**79.54**	65.30	78.88
eu	U	87.65	85.56ϟ	93.54	**95.54**	88.12	95.52
	R	70.26ϟ	71.21ϟ	54.07	**72.99**	59.47	74.14
fr	U	94.81	95.62ϟ	98.09	**98.42**	96.21	98.45
	R	80.12ϟ	85.55ϟ	78.08	**88.60**	80.05	88.55
he	U	92.47	92.05ϟ	95.24	**97.88**	85.34	97.52
	R	84.43	84.54	90.98	**95.36**	69.11	95.36
hu	U	77.75	75.71	95.08	**95.81**	76.42	96.28
	R	**75.63**ϟ	71.05ϟ	61.17	74.28	60.05	75.71
ko	U	79.89	80.54ϟ	91.14	**94.67**	65.05	94.49
	X	66.57	67.10ϟ	81.40	**85.33**	41.17	85.67
	R	65.24ϟ	65.52ϟ	46.65	**66.56**	33.29	66.42
pl	U	90.35	91.09ϟ	97.53	**98.82**	90.84	98.86
	X	76.23	79.47ϟ	92.58	**96.13**	71.47	95.54
	R	64.70ϟ	64.34ϟ	48.55	**66.10**	53.52	66.92
sv	U	88.74	90.80ϟ	97.67	**98.18**	87.59	98.22
	X	83.39	82.32ϟ	92.14	**95.89**	78.58	95.87
	R	80.51	80.08ϟ	90.24	**94.38**	77.19	94.61
μ	U	87.70	87.86	95.61	**96.94**	87.34	97.02
	X	79.05	80.29	90.19	**93.69**	70.97	93.69
	R	73.71	74.22	66.87	**79.73**	62.25	80.07

Table 1: PoS tagging accuracy. Each subrow shows the prediction of a different tag: universal (U) or language-specific (X) for UD; and rich (R) for SPMRL. **DiT**, **GaT** and **SSD** stand for the diffusion, adversarial and SSD-based models. The **LSTM** column indicates the best undirectional (first subcolumn) and bidirectional (second subcolumn) LSTM-based model. The ϟ symbol indicates whether the best result comes from the xLSTM. Language acronyms from ISO-639. **Bold** for the best non-baseline, underline for the best SL model. Average across SPMRL and UD treebanks in the last block (μ).

Table 2 shows the mention-level F1 score on the NER datasets, where the adversarial tagger offers competitive performance against the baseline model, reaching the highest score in 6 out of 12 datasets, while MAMBA-2 achieves the best result on DrugsNER. We hypothesize that due to its reliance on MSE loss, the diffusion tagger underperforms on highly unbalanced datasets like Weibo, failing to surpass even basic non-pretrained models.

	LSTM		**DiT**	**GaT**	**SSD**	**Base**
CoNLL$^{35}_{en}$	51.62ϟ	56.03ϟ	65.57	**72.63**	64.64	71.79
BioNLP$^{39}_{en}$	40.90	43.88ϟ	51.62	**63.05**	53.97	63.48
DrugsNER$^{25}_{en}$	98.40	98.35ϟ	94.88	98.39	**98.44**	98.46
EIEC$^{26}_{eu}$	17.68	14.72ϟ	34.42	**43.61**	28.62	42.80
GermEval$^{16}_{de}$	30.64	28.54	44.66	**50.03**	41.95	50.16
HiNER$^{27}_{hi}$	67.85	71.88ϟ	69.80	**75.25**	53.43	75.30
JNLPBA$^{40}_{en}$	44.42	54.46	48.46	**62.05**	49.30	61.99
NYTK$^{28}_{hu}$	39.46	37.27ϟ	71.49	**76.32**	62.80	76.36
Webbnyh.$^{14}_{sv}$	20.58	19.36	29.35	**31.39**	28.02	32.58
Weibo$^{14}_{zh}$	21.40ϟ	26.88ϟ	19.59	**42.29**	24.65	38.58
WikiNER$^{37}_{fr}$	81.71ϟ	85.59ϟ	89.72	**92.69**	87.32	92.63
WikiNER$^{47}_{pl}$	82.15	84.95	86.63	**93.63**	87.10	93.47
μ	49.73	51.82	58.84	**66.78**	56.68	66.46

Table 2: Mention-level F1 score. To compare the label set balance, the Shannon entropy is annotated in superscripts. Same notation as in Table 1.

Tables 3, 4 and 5 focus on constituent, dependency and graph parsing. In contrast to structurally simpler tasks (where MAMBA-2 offered fair performance) our results suggest that structured state-space models struggle to capture token dependencies within the input sentence, performing on par with the left-to-right (x)LSTM encoder and lagging behind all pretrained models. While the diffusion model achieves relatively good results, it still falls short of the baseline, especially for graph encodings (74.63 vs. 86.33 average LF) under low-resource settings (12.44 vs. 63.99 LF on Tamil). The adversarial tagger slightly improves over the baseline in dependency and constituency parsing: 87.70 vs. 87.49 LAS and 91.04 vs. 90.77 LF, respectively; but matches the baseline in graph parsing (86.27 vs. 86.33 LF).

To assess the gains of the adversarial tagger, we measured the ratio of well-formed label sequences[8] on the PTB constituency treebank (chosen due to space reasons and its widespread use). The adversarial tagger produced valid label sequences for 82.13% of the test set, compared to 80.76% for the baseline. This likely stems from the adversarial loss, which pushes the tagger to produce sequences that fool the discriminator, improving performance.

[8] We consider a label sequence to be well-formed if it corresponds to the encoding of a tree, that is, if a valid tree is obtained directly from the decoding process directly, without the need of postprocessing to deal with issues like undefined nonterminals or out-of-range indexes.

	LSTM		DiT	GaT	SSD	Base	Span
PTB	$59.36^{\lightning}_{\mathrm{R}}$	$70.68^{\lightning}_{\mathrm{R}}$	92.76_{R}	$\underline{\mathbf{94.96}}_{\mathrm{R}}$	63.16_{R}	94.88_{R}	95.59
CTB	$56.33^{\lightning}_{\mathrm{T}}$	$79.64^{\lightning}_{\mathrm{R}}$	87.87_{T}	$\underline{\mathbf{93.52}}_{\mathrm{T}}$	57.70_{T}	90.84_{R}	91.75
de	$40.36^{\lightning}_{\mathrm{T}}$	$77.29^{\lightning}_{\mathrm{T}}$	88.05_{R}	$\mathbf{91.26}_{\mathrm{T}}$	40.49_{R}	$\underline{92.12}_{\mathrm{T}}$	90.20
eu	$55.81^{\lightning}_{\mathrm{T}}$	$74.62^{\lightning}_{\mathrm{T}}$	80.20_{R}	$\mathbf{89.24}_{\mathrm{R}}$	48.55_{R}	$\underline{89.29}_{\mathrm{T}}$	91.63
fr	$46.66^{\lightning}_{\mathrm{T}}$	$72.63^{\lightning}_{\mathrm{T}}$	84.91_{R}	$\underline{\mathbf{86.52}}_{\mathrm{R}}$	47.44_{R}	84.79_{T}	87.42
he	79.53_{T}	$85.25^{\lightning}_{\mathrm{T}}$	91.17_{R}	$\underline{\mathbf{92.47}}_{\mathrm{R}}$	61.61_{R}	92.43_{R}	92.99
hu	60.32_{R}	$71.83^{\lightning}_{\mathrm{T}}$	89.40_{R}	$\mathbf{92.42}_{\mathrm{R}}$	55.78_{R}	$\underline{92.47}_{\mathrm{R}}$	94.90
ko	$56.32^{\lightning}_{\mathrm{T}}$	$71.00^{\lightning}_{\mathrm{T}}$	85.18_{R}	$\underline{\mathbf{87.60}}_{\mathrm{R}}$	41.18_{R}	86.93_{T}	88.80
pl	$73.19^{\lightning}_{\mathrm{T}}$	$89.47^{\lightning}_{\mathrm{T}}$	94.07_{T}	$\mathbf{94.39}_{\mathrm{R}}$	62.77_{R}	$\underline{95.85}_{\mathrm{T}}$	96.36
sv	48.81_{R}	$63.03^{\lightning}_{\mathrm{R}}$	81.26_{R}	$\mathbf{88.02}_{\mathrm{R}}$	44.68_{R}	$\underline{88.12}_{\mathrm{R}}$	88.87
μ	57.67	75.54	87.59	$\underline{\mathbf{91.04}}$	52.24	90.77	91.85

Table 3: LF score for constituency parsers. Same notation as in Table 1. Only the best encoding is displayed in subscripts: relative (**R**) and tetra-tagging (**T**). The SL (**Base**) and span-based (**Span**) baselines are included in the last columns.

	LSTM		DiT	GaT	SSD	Base	Biaf
PTB	$62.65^{\lightning}_{\mathrm{B7}}$	$90.41^{\lightning}_{\mathrm{B}}$	92.39_{B7}	$\mathbf{94.19}_{\mathrm{B7}}$	66.25_{B}	$\underline{94.34}_{\mathrm{B4}}$	95.36
CTB	$50.41^{\lightning}_{\mathrm{B}}$	$82.18^{\lightning}_{\mathrm{B}}$	87.93_{B4}	$\underline{\mathbf{89.19}}_{\mathrm{B4}}$	44.78_{B7}	88.63_{B4}	89.67
de	$60.81^{\lightning}_{\mathrm{B}}$	$76.07^{\lightning}_{\mathrm{B7}}$	80.75_{B4}	$\underline{\mathbf{83.14}}_{\mathrm{B7}}$	52.34_{B}	82.77_{B7}	85.59
eu	$45.48^{\lightning}_{\mathrm{B}}$	$68.44^{\lightning}_{\mathrm{B7}}$	79.43_{B4}	$\mathbf{82.54}_{\mathrm{B7}}$	31.34_{B4}	$\underline{82.58}_{\mathrm{B7}}$	87.35
fr	$69.62^{\lightning}_{\mathrm{B}}$	$84.90^{\lightning}_{\mathrm{H}}$	90.78_{H}	$\underline{\mathbf{91.99}}_{\mathrm{B}}$	67.39_{B7}	91.59_{B7}	94.14
he	$66.75^{\lightning}_{\mathrm{B}}$	$80.31^{\lightning}_{\mathrm{B7}}$	86.43_{B7}	$\underline{\mathbf{89.55}}_{\mathrm{B4}}$	49.97_{B7}	89.26_{H}	91.34
hu	$43.54^{\lightning}_{\mathrm{B}}$	$55.05^{\lightning}_{\mathrm{B4}}$	72.16_{B4}	$\mathbf{79.51}_{\mathrm{B4}}$	23.19_{B4}	$\underline{80.12}_{\mathrm{B7}}$	86.46
ko	$55.04^{\lightning}_{\mathrm{B}}$	$72.85^{\lightning}_{\mathrm{B4}}$	81.52_{H}	$\underline{\mathbf{84.28}}_{\mathrm{B}}$	22.52_{B7}	83.63_{B}	83.07
pl	$64.15^{\lightning}_{\mathrm{B}}$	$81.04^{\lightning}_{\mathrm{B4}}$	89.30_{B7}	$\underline{\mathbf{91.01}}_{\mathrm{B}}$	52.98_{B}	90.76_{B7}	93.85
sv	$60.32^{\lightning}_{\mathrm{B4}}$	$78.06^{\lightning}_{\mathrm{B4}}$	89.54_{B4}	$\underline{\mathbf{91.60}}_{\mathrm{B7}}$	40.06_{B7}	91.20_{B7}	91.76
μ	57.88	76.93	85.02	$\mathbf{87.70}$	45.08	87.49	89.85

Table 4: LAS score for dependency parsers. Same notation as in Table 3 with acronyms: 2-planar bracketing (**B**), 4-bit (**B4**), 7-bit (**B7**), hexa-tagging (**H**). The SL (**Base**) and biaffine (**Biaf**) baselines are included in the last columns.

	LSTM		DiT	GaT	SSD	Base	Biaf
en	$65.30^{\lightning}_{\mathrm{R}}$	$86.70^{\lightning}_{\mathrm{B}}$	85.91_{B}	$\underline{\mathbf{93.57}}_{\mathrm{B}}$	64.37_{R}	93.48_{B}	94.15
zh	$46.87^{\lightning}_{\mathrm{R}}$	$76.03^{\lightning}_{6k}$	64.09_{6k}	$\mathbf{83.00}_{6k}$	37.64_{R}	$\underline{83.23}_{6k}$	88.91
ar	$67.86^{\lightning}_{\mathrm{R}}$	$73.31^{\lightning}_{6k}$	76.61_{6k}	$\underline{\mathbf{81.75}}_{6k}$	57.09_{R}	81.08_{B}	84.74
bg	$67.78^{\lightning}_{6k}$	$82.78^{\lightning}_{6k}$	86.83_{6k}	$\mathbf{90.47}_{4k}$	50.31_{6k}	$\underline{90.64}_{4k}$	93.57
cs	$59.82^{\lightning}_{6k}$	$78.95^{\lightning}_{\mathrm{R}}$	82.27_{4k}	$\underline{\mathbf{88.45}}_{6k}$	53.92_{6k}	88.23_{6k}	89.79
fr	$70.95^{\lightning}_{6k}$	$82.21^{\lightning}_{6k}$	83.29_{6k}	$\underline{\mathbf{91.73}}_{6k}$	66.00_{6k}	90.97_{6k}	95.10
it	$72.83^{\lightning}_{6k}$	$86.69^{\lightning}_{6k}$	87.85_{6k}	$\mathbf{91.46}_{4k}$	69.46_{6k}	$\underline{91.87}_{6k}$	94.00
nl	$59.30^{\lightning}_{6k}$	$77.19^{\lightning}_{\mathrm{R}}$	81.70_{6k}	$\mathbf{87.75}_{6k}$	48.74_{6k}	$\underline{87.88}_{6k}$	94.38
pl	$64.06^{\lightning}_{6k}$	$81.38^{\lightning}_{\mathrm{R}}$	85.27_{6k}	$\mathbf{91.76}_{6k}$	54.17_{6k}	$\underline{91.97}_{6k}$	92.08
ta	$40.49^{\lightning}_{6k}$	$52.21^{\lightning}_{4k}$	12.44_{B}	$\mathbf{62.79}_{6k}$	15.81_{R}	$\underline{63.99}_{6k}$	92.06
μ	61.53	77.74	74.63	$\mathbf{86.27}$	51.75	$\underline{86.33}$	91.87

Table 5: LF score for graph parsers. Same notation as in Tables 3 and 4, with acronyms: relative (**R**), bracketing with $k = 3$ (**B**), $4k$-bit with $k = 4$ (**4*k***) and $6k$-bit with $k = 4$ (**6*k***) encodings. The SL (**Base**) and biaffine (**Biaf**) baselines are included in the last columns.

The effect weakens in graph parsing, where label sequences are less constrained, as graphs are more expressive and need not be connected or acyclic.

Speed analysis Figure 1 compares the performance vs speed of our dependency parsers. Unidirectional encoders (LSTM, xLSTM, MAMBA) although performing well on tagging, suffer considerably in parsing. The baseline and adversarial tagger pair in speed since at inference time both rely on the same number of parameters. The BiLSTM and BixLSTM, although falling behind the pretrained models by ∼4 points, offer a fast yet accurate approach.[9]

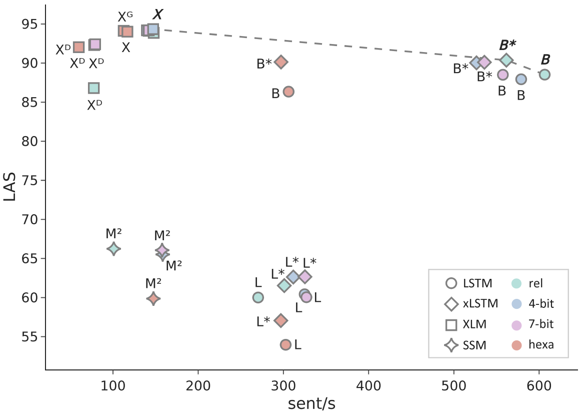


Figure 1: Pareto front of LAS vs. speed (sent/s) on PTB dependency parsing. Colors are reserved for encodings, symbols and text annotations for architectures: LSTM (**L**), BiLSTM (**B**), xLSTM (**L***), BixLSTM (**B***), MAMBA-2 (**M^2**), XLM (**X**), DiT (**X^D**) and GaT (**X^G**). All executions were measured on the same NVIDIA RTX 3090 24GB.

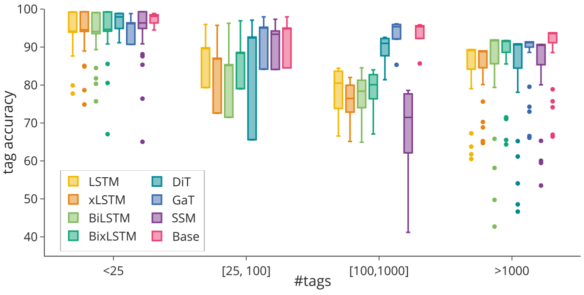


Figure 2: PoS and NER accuracy (y) across output spaces (x, uneven intervals).

Impact of label space Figure 2 shows the performance consistency of each architecture as the label space increases in PoS tagging and NER tasks. In

[9] In our LSTM-based encoder, each layer is a LSTM unit with hidden size $d = 400$. For their bidirectional counterpart each layer has two units—each processing the input in a different directions—with $d = 200$, maintaining the overall layer size to 400. Under this configuration the complexity of the unidirectional encoder scales with $d = 400$ while the bidirectional with $d = 200$.

general, all models experience a degradation in performance as the output space grows, particularly non-Transformer models (LSTM- and SSD-based), which exhibit greater performance variability even in smaller label spaces. Among non-baselines, the adversarial tagger is most stable, with few outliers even under large label spaces, and performance matching the baseline.

Parsing performance Figure 3 provides a fine-grained analysis of parsing performance on the PTB dataset, specifically evaluating F1-scores relative to constituent span length and dependency displacement.[10] Broadly, unidirectional models (LSTMs and state-space models) exhibit a substantial performance decay when modeling long-range dependencies compared to their bidirectional counterparts. This trend aligns with recent findings on the challenges of incrementality (Ezquerro et al., 2023, 2024a), highlighting a persistent limitation that remains unresolved even in large pretrained architectures like MAMBA. Additionally, while the diffusion tagger shows promise on simpler tagging problems (Table 1), it barely outperforms Bi(x)LSTM encoders on parsing tasks, despite relying on a powerful Transformer-based encoder. This marginal gain suggests that diffusion tagging still struggles to effectively internalize hierarchical information as a sequence of tags.

6 Conclusion

In this work we study alternative architectures for sequence labeling, examining models such as diffusion and adversarial approaches, which fully redefine how the label space is modeled, and non-Transformer contextualizers like xLSTMs and SSMs, which modify the underlying backbone for sequence modeling. Our unified evaluation reveals that both lines of work can rival or even surpass standard Transformer-based setups on simple tasks such as PoS tagging and NER. However, when considering more complex tasks that demand long-range dependencies, only the adversarial tagger maintains competitive performance against traditional methods. Its ability to exploit generative modeling proves especially effective in capturing well-formed structures beyond the capabilities of other non-standard architectures.

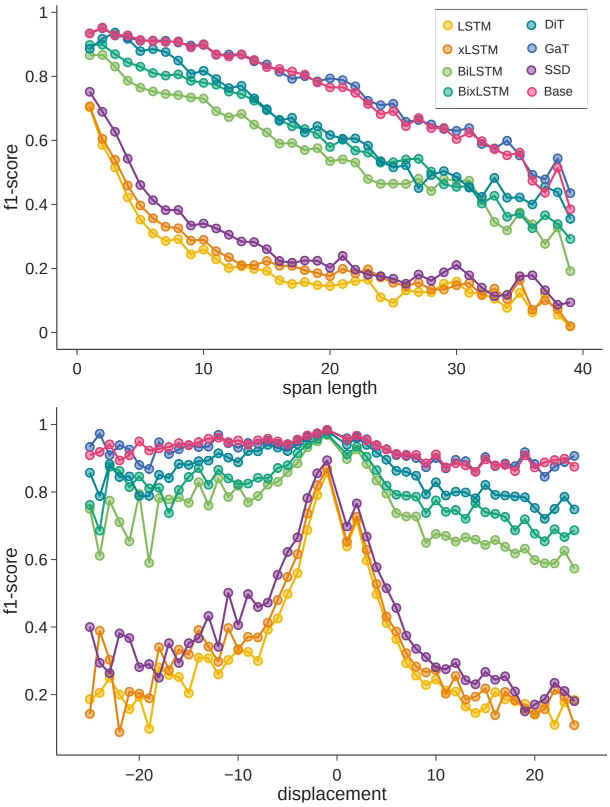


Figure 3: Parsing performance on the PTB test set. The top panel shows F1-scores relative to span length (constituent parsing), while the bottom panel shows performance relative to dependency displacement (dependency parsing).

Limitations

Physical resources Our computational resources consists of 8 24GB RTX 3090 and 3 40GB A100 GPUs. We also have limited access to a large computing infrastructure with more than 300 nodes, where each node contains 8 40GB A100.

Lack of generative models as encoders For models that include Transformers as components, we rely on masked language models (MLMs) rather than generative models. The first reason is that incremental sequence tagging with autoregressive models lags behind bidirectional encoders, as recently shown by Ezquerro et al. (2023, 2024a). Given this, our choice of masked language models aligns with current best practices for sequence labeling. The second reason is computational constraints. While our setup allowed extensive experimentation, training large-scale generative models would require significantly higher resources. As such, our focus remains on models that are both effective and computationally accessible.

[10] Following Anderson and Gómez-Rodríguez (2020), we define displacement as the signed distance $d - h$ between the dependent (d) and head (h) positions.

References

Iñaki Alegria, Olatz Arregi, Irene Balza, Nerea Ezeiza, Izaskun Fernandez, and Ruben Urizar. 2004. Design and Development of a Named Entity Recognizer for an Agglutinative Language. In *First International Joint Conference on NLP (IJC NLP04), Workshop on Named Entity Recognition*.

Afra Amini and Ryan Cotterell. 2022. On parsing as tagging. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8884–8900, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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.

Mark Anderson and Carlos Gómez-Rodríguez. 2020. Inherent Dependency Displacement Bias of Transition-Based Algorithms. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5147–5155, Marseille, France. European Language Resources Association.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. 2024. xLSTM: Extended Long Short-Term Memory. *Preprint*, arXiv:2405.04517.

Darina Benikova, Chris Biemann, Max Kisselew, and Sebastian Padó. 2014. GermEval 2014 Named Entity Recognition Shared Task: Companion Paper. In *Proceedings of the KONVENS GermEval workshop*, pages 104–112, Hildesheim, Germany.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2021. From Raw Text to Enhanced Universal Dependencies: The Parsing Shared Task at IWPT 2021. 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 146–157, Online. Association for Computational Linguistics.

Nigel Collier and Jin-Dong Kim. 2004. Introduction to the Bio-entity Recognition Task at JNLPBA. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)*, pages 73–78, Geneva, Switzerland. COLING.

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10041–10071. PMLR.

Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *International Conference on Learning Representations*.

Ana Ezquerro, Carlos Gómez-Rodríguez, and David Vilares. 2023. On the Challenges of Fully Incremental Neural Dependency Parsing. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 52–66, Nusa Dua, Bali. Association for Computational Linguistics.

Ana Ezquerro, Carlos Gómez-Rodríguez, and David Vilares. 2024a. From partial to strictly incremental constituent parsing. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 225–233, St. Julian's, Malta. Association for Computational Linguistics.

Ana Ezquerro, David Vilares, and Carlos Gómez-Rodríguez. 2024b. Dependency graph parsing as sequence labeling. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11804–11828, Miami, Florida, USA. Association for Computational Linguistics.

Zhujin Gao, Junliang Guo, Xu Tan, Yongxin Zhu, Fang Zhang, Jiang Bian, and Linli Xu. 2024. Empowering Diffusion Models on the Embedding Space for Text Generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4664–4683, Mexico City, Mexico. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Diego Roca, and David Vilares. 2023. 4 and 7-bit Labeling for Projective and Non-Projective Dependency Trees. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6375–6384, Singapore. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium. Association for Computational Linguistics.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

A. Graves and J. Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4.

Albert Gu and Tri Dao. 2023. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *Preprint*, arXiv:2312.00752.

Albert Gu, Karan Goel, and Christopher Re. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*.

Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2004. Semantic Role Labeling by Tagging Syntactic Chunks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*, pages 110–113, Boston, Massachusetts, USA. Association for Computational Linguistics.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. 2023. SSD-LM: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11575–11596, Toronto, Canada. Association for Computational Linguistics.

Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. 2023. DiffusionBERT: Improving Generative Masked Language Models with Diffusion Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4521–4534, Toronto, Canada. Association for Computational Linguistics.

Moein Heidari, Ehsan Khodapanah Aghdam, Alexander Manzella, Daniel Hsu, Rebecca Scalabrino, Wenjin Chen, David J Foran, and Ilker Hacihaliloglu. 2025. A study on the performance of u-net modifications in retroperitoneal tumor segmentation. *arXiv preprint arXiv:2502.00314*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

S Hochreiter and J Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Ziyang Huang, Pengfei Cao, Jun Zhao, and Kang Liu. 2023. DiffusionSL: Sequence labeling via tag diffusion process. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12902–12920, Singapore. Association for Computational Linguistics.

HuggingFace. 2024. pnr-svc/drugs-ner-data.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual Constituency Parsing with Self-Attention and Pre-Training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2020. Tetra-Tagging: Word-Synchronous Parsing with Linear-Time Inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6255–6261, Online. Association for Computational Linguistics.

Matt J. Kusner and José Miguel Hernández-Lobato. 2016. GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution. *arXiv e-prints*, arXiv:1611.04051.

John Lafferty, Andrew McCallum, Fernando Pereira, et al. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Icml*, volume 1, page 3. Williamstown, MA.

Yann LeCun, Yoshua Bengio, et al. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

Yangming Li, Han Li, Kaisheng Yao, and Xiaolong Li. 2020. Handling rare entities for neural sequence labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6441–6451, Online. Association for Computational Linguistics.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Yuhan Liu, Shangbin Feng, Xiaochuang Han, Vidhisha Balachandran, Chan Young Park, Sachin Kumar, and Yulia Tsvetkov. 2024. P^{3}Sum: Preserving author's perspective in news summarization with diffusion language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2154–2173, Mexico City, Mexico. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

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.

Rudra Murthy, Pallab Bhattacharjee, Rahul Sharnagat, Jyotsana Khatri, Diptesh Kanojia, and Pushpak Bhattacharyya. 2022. HiNER: A large Hindi Named Entity Recognition Dataset. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4467–4476, Marseille, France. European Language Resources Association.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

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.

Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. 2012. Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, 194:151–175.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado. Association for Computational Linguistics.

Sungjoon Park, Jihyung Moon, Sungdong Kim, Won Ik Cho, Ji Yoon Han, Jangwon Park, Chisung Song, Junseong Kim, Youngsook Song, Taehwan Oh, Joohong Lee, Juhyun Oh, Sungwon Lyu, Younghoon Jeong, Inkwon Lee, Sangwoo Seo, Dongjun Lee, Hyunwoo Kim, Myeonghwa Lee, Seongbo Jang, Seungwon Do, Sunkyoung Kim, Kyungtae Lim, Jongwon Lee, Kyumin Park, Jamin Shin, Seonghyun Kim, Lucy Park, Lucy Park, Alice Oh, Jung-Woo Ha (NAVER AI Lab), Kyunghyun Cho, and Kyunghyun Cho. 2021. KLUE: Korean Language Understanding Evaluation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Kevin Parnow, Zuchao Li, and Hai Zhao. 2021. Grammatical Error Correction as GAN-like Sequence Labeling. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3284–3290, Online. Association for Computational Linguistics.

Nanyun Peng and Mark Dredze. 2015. Named Entity Recognition for Chinese Social Media with Jointly Trained Embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 548–554, Lisbon, Portugal. Association for Computational Linguistics.

Sampo Pyysalo, Tomoko Ohta, and Sophia Ananiadou. 2013. Overview of the Cancer Genetics (CG) task of BioNLP Shared Task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 58–66, Sofia, Bulgaria. Association for Computational Linguistics.

Lance Ramshaw and Mitch Marcus. 1995. Text Chunking using Transformation-Based Learning. In *Third Workshop on Very Large Corpora*.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. DiffusionNER: Boundary diffusion for named entity recognition. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3875–3890, Toronto, Canada. Association for Computational Linguistics.

Wenkai Shi, Wenbin An, Feng Tian, Qinghua Zheng, QianYing Wang, and Ping Chen. 2023. A Diffusion Weighted Graph Framework for New Intent Discovery. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8033–8042, Singapore. Association for Computational Linguistics.

Eszter Simon and Noémi Vadász. 2021. Introducing NYTK-NerKor, A Gold Standard Hungarian Named Entity Annotated Corpus. In *Text, Speech, and Dialogue: 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6–9, 2021, Proceedings*, page 222–234, Berlin, Heidelberg. Springer-Verlag.

Jiaming Song, Chenlin Meng, and Stefano Ermon. 2021. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*.

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

Språkbanken Text. 2024. Webbnyheter 2012.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling.

In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium. Association for Computational Linguistics.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable dependency parsing as sequence labeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 717–723, Minneapolis, Minnesota. Association for Computational Linguistics.

Yan Sun, Yutong Lu, Yan Yi Li, Zihao Jing, Carson K. Leung, and Pingzhao Hu. 2025. Molgraph-xlstm: A graph-based dual-level xlstm framework with multi-head mixture-of-experts for enhanced molecular representation and interpretability. *Preprint*, arXiv:2501.18439.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Yu Tong, Ge Chen, Guokai Zheng, Rui Li, and Jiang Dazhi. 2024. When Generative Adversarial Networks Meet Sequence Labeling Challenges. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10625–10635, Miami, Florida, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

David Vilares and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2771–2785, Online. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.*, 11(2):207–238.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Yue Zhang and Stephen Clark. 2008. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL-08: HLT*, pages 888–896, Columbus, Ohio. Association for Computational Linguistics.

Kun Zhou, Yifan Li, Xin Zhao, and Ji-Rong Wen. 2024. Diffusion-NAT: Self-Prompting Discrete Diffusion for Non-Autoregressive Text Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1438–1451, St. Julian's, Malta. Association for Computational Linguistics.

A Visualizations

Figure 4 illustrates the forward and denoising process of the diffusion tagger. Figure 5 shows the loss flow of the adversarial training.

B Treebank statistics

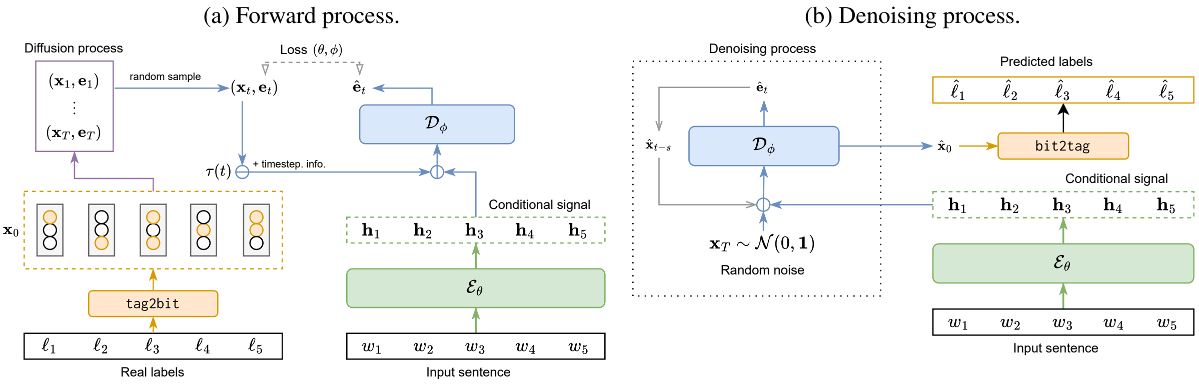
Table 6 summarizes the number of tags required in our datasets. We selected the following UD treebanks to train and evaluate our dependency parsers: German GSD, Basque BDT, French GSD, Hebrew HTB, Hungarian Szeged, Korean KAIST, Polish PDB, and Swedish Talbanken. For graph parsing, we drew from the SDP (Oepen et al., 2015) and IWPT (Bouma et al., 2021) datasets, selecting the English DM, Chinese PAS, Arabic PADT, Bulgarian BTB, Czech PSD, French Sequoia, Italian IST, Dutch Alpino, Polish PDB, and Tamil TTB treebanks. These languages were chosen to reflect a broad range of syntactic structures and to overlap with those in the SPMRL corpus, enabling more consistent cross-task comparisons of tagging performance in typologically similar languages.

C Existing parsing linearizations

In our empirical study, we relied on existing sequence-labeling approaches to cast tree- or graph-structured prediction tasks in terms of token-level classification. In this section, we provide a detailed description of each encoding, along with illustrative examples to clarify the mapping from the original structured input to its token-level representation. We also highlight the advantages and potential limitations of each approach in preserving critical structural dependencies.

C.1 Constituency parsing

For our study, we relied on the relative (Gómez-Rodríguez and Vilares, 2018) and tetratagging (Kitaev and Klein, 2020) linearizations. Figure 6 illustrates a constituent tree and its transformation into label sequences under each encoding scheme.


(a) Forward process.

(b) Denoising process.

Diffusion process
$(\mathbf{x}_1, \mathbf{e}_1)$
$(\mathbf{x}_T, \mathbf{e}_T)$
random sample
$(\mathbf{x}_t, \mathbf{e}_t)$
Loss (θ, ϕ)
$\hat{\mathbf{e}}_t$
$\mathcal{D}_\phi$
$\tau(t)$
+ timestep. info.
$\mathbf{x}_0$
Conditional signal
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$ $\mathbf{h}_5$
$\mathcal{E}_\theta$
tag2bit
ℓ_1 ℓ_2 ℓ_3 ℓ_4 ℓ_5
Real labels
w_1 w_2 w_3 w_4 w_5
Input sentence
Denoising process
$\hat{\mathbf{e}}_t$
$\hat{\mathbf{x}}_{t-s}$
$\mathcal{D}_\phi$
$\hat{\mathbf{x}}_0$
$\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{1})$
Random noise
Predicted labels
$\hat{\ell}_1$ $\hat{\ell}_2$ $\hat{\ell}_3$ $\hat{\ell}_4$ $\hat{\ell}_5$
bit2tag
Conditional signal
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$ $\mathbf{h}_5$
$\mathcal{E}_\theta$
w_1 w_2 w_3 w_4 w_5
Input sentence

Figure 4: Diffusion tagger in forward and denoising steps. The symbol ⦶ is the concatenation operator and an open arrow (⇗) loss propagation. In Figure 4a, $\mathcal{E}_\theta$ embeds the sentence as the conditional signal. The real labels are transformed into bits and fed to the diffusion process, where the latent $\mathbf{x}_t$ is computed from the sampled noise $\mathbf{e}_t$, and concatenated with time embeddings $\tau(t)$ and the conditional signal. Then, $\mathcal{D}_\phi$ learns to extract the noise that was added to $\mathbf{x}_t$. All parameters are optimized with the MSE loss between the real and predicted noise. Figure 4b shows the denoising process. The conditional signal is computed once with $\mathcal{E}_\theta$ and an initial signal $\mathbf{x}_T$ is sampled from Gaussian noise. Iteratively, $\mathcal{D}_\phi$ removes noise from the input and conditional signal and estimates the previous latent $\hat{\mathbf{x}}_{t-s}$ until $\hat{\mathbf{x}}_0$ is reached. Then, $\hat{\mathbf{x}}_0$ is fed to the BT module to recover a sequence of predicted labels.

	Dep.						Cons.	
	UPOS	XPOS	REL	B	B4	B7	POS	R
PTB	45	45	45	235	16	22	46	39
CTB	36	36	19	337	16	16	42	26
de	17	52	45	311	16	69	30k	16
eu	17	-	33	250	16	81	30k	16
fr	16	2	56	235	16	51	39k	28
he	15	-	37	164	16	37	281	33
hu	16	-	52	196	16	61	50k	19
ko	17	2k	32	150	16	40	105k	25
pl	17	856	67	246	16	71	27k	21
sv	17	144	44	176	16	46	357	23

	Graph			
	R	B	$4k$	$6k$
en	8139	506	710	650
zh	28567	1315	1001	859
ar	2665	1076	448	466
bg	917	522	268	239
cs	5200	2225	1179	1139
fr	968	578	229	208
it	2004	900	306	285
nl	1093	884	371	321
pl	2380	1541	679	598
ta	194	118	74	45

CoNLL	BioNLP	DrugsNER	EIEC	GermEval	HiNER
45	52	1611	9	25	23

JNLPBA	NYTK	Webbnyh.	Weibo	WikiNER
11	9	5	17	4

Table 6: Number of learned tags in our experiments for PoS tagging; dependency, constituency and graph parsing; and NER. Languages are specified with the ISO-639 code and the corpus is specified in subscripts. Encodings are abbreviated as in Tables 3, 4 and 5: bracketing (**B**), 4-bit (**B4**) and 7-bit (**B7**) encodings for dependency parsing; relative (**R**) for constituency parsing; and relative (**R**), bracketing with $k = 3$ (**B**), $4k$-bit with $k = 4$ (**4*k***) and $6k$-bit with $k = 4$ (**6*k***). **UPOS**, **XPOS** and **REL** refer to the number of unique tags in the CoNLL format; and **POS** indicates the number of tags of the PTB, CTB and SPMRL corpus.

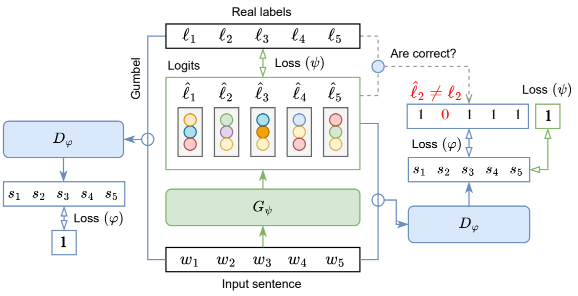

Real labels
ℓ_1 ℓ_2 ℓ_3 ℓ_4 ℓ_5
Gumbel
Logits
Loss (ψ)
Are correct?
$\hat{\ell}_1$ $\hat{\ell}_2$ $\hat{\ell}_3$ $\hat{\ell}_4$ $\hat{\ell}_5$
$\hat{\ell}_2 \neq \ell_2$
Loss (ψ)
1 0 1 1 1
1
Loss (φ)
s_1 s_2 s_3 s_4 s_5
D_φ
s_1 s_2 s_3 s_4 s_5
Loss (φ)
1
G_ψ
D_φ
w_1 w_2 w_3 w_4 w_5
Input sentence

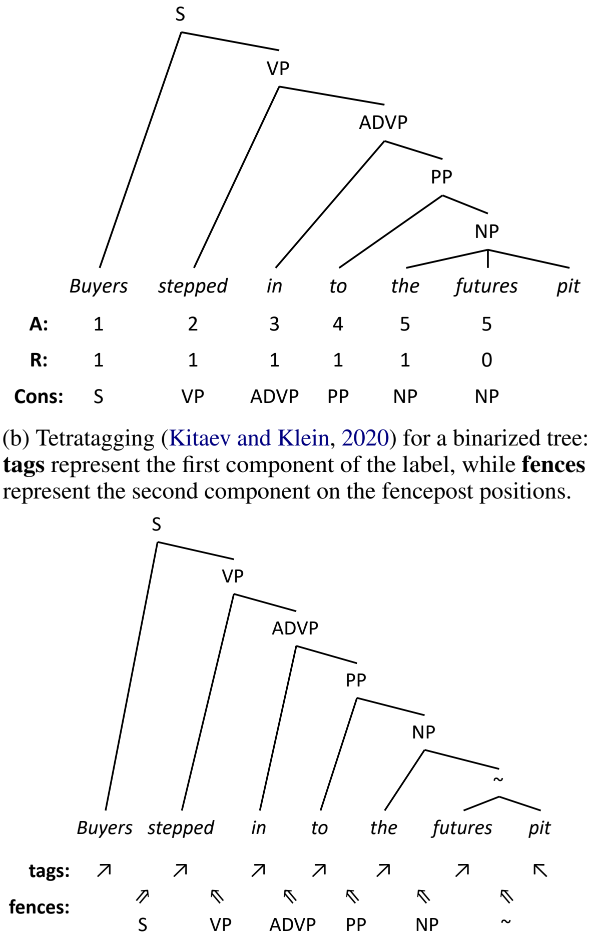
Figure 5: Adversarial tagger view (symbols as in Figure 4). G_ψ (green) is trained with the tag loss. D_φ (blue) learns to distinguish valid tag sequences and guides G_ψ.

Relative encoding Given a constituent tree with no unary chains[11] over the sentence $(w_1 \cdots w_n)$, the absolute encoding proposed by Gómez-Rodríguez and Vilares (2018) represents an input tree with a sequence of $n-1$ labels, where each label consists of two components $\ell_i^A = (p_i, c_i)$, for $i = 1, ..., n-1$. The first component $p_i \in \mathbb{Z}^+$ indicates the number of constituents shared between w_i and w_{i+1}; while the second component c_i is the lowest shared constituent. The relative encoding is directly obtained from the absolute encoding by building each label as $\ell_i^{\mathrm{R}} = (p_i - p_{i-1}, c_i)$ where $i > 1$; otherwise $\ell_i^{\mathrm{R}} = \ell_i^{\mathrm{A}}$.

The relative decoding only requires processing the tree from left to right, opening intermediate

[11] As proposed by Gómez-Rodríguez and Vilares (2018), we remove unary chains by collapsing their constituents into a single node (e.g., the chain S-NP is collapsed into a single node with the constituent S:NP).

(a) Absolute (**A**) and relative (**R**) encodings (Gómez-Rodríguez and Vilares, 2018). The row **Cons.** is the second component of the label and remains the same for both variants.



(b) Tetratagging (Kitaev and Klein, 2020) for a binarized tree: **tags** represent the first component of the label, while **fences** represent the second component on the fencepost positions.

Figure 6: Example of a constituent tree encoded with the relative encoding (Figure 6a) and tetratagging (Figure 6b).

nodes when upon encountering a positive value in the first component of each label and resolving them when finding negative values or reaching the end of the sequence.

Tetratagging Kitaev and Klein (2020) encode a binary constituent tree with n labels, where each label $\ell_i^{\mathrm{T}} = (t_i, f_i, c_i)$ is composed of three components, and the last one is always defined as $\ell_n^{\mathrm{T}} = (\nwarrow, \emptyset, \emptyset)$. The first component of the label t_i represents with an arrow symbol whether the terminal node w_i is a left ($\nearrow$) or right descendant ($\nwarrow$) of its parent. The second and third component, f_i and c_i, also determine whether the lowest non-terminal node that covers the fencepost between w_i and w_{i+1} is a left ($\Nearrow$) or right ($\Nwarrow$) child and the constituent of the lowest common node.

C.2 Dependency parsing

Given a dependency tree $G = (W, A)$, where $W = (w_1 \cdots w_n) \in \mathcal{V}^n$ is the input sentence, and $A = \{(h \xrightarrow{r} d) : d = 1 \cdots n; h \in [0, n], h \neq d, r \in \mathcal{R}\}$[12] is the arc set, a tree linearization represents the information of A as a sequence of n labels $(\ell_1 \cdots \ell_n) \in \mathcal{L}^n$. Figure 7 shows two examples of a dependency tree encoded with the 4-bit and 7-bit encodings (Gómez-Rodríguez et al., 2023) and hexatagging (Amini et al., 2023).

Projectivity and planarity A dependency tree G is a connected, acyclic labeled graph where each node has only one incoming arc. We say that a dependency tree is *projective* when no arcs of A crosses each other. For instance, the tree in Figure 7a is projective, while the tree in Figure 7c is non-projective because the arc $(4 \xrightarrow{\text{case}} 7)$ crosses $(3 \xrightarrow{\text{acl:relcl}} 6)$.

Assuming a non-projective dependency tree, the arcs of A can be distributed into at least k mutually exclusive subsets (also denoted as *planes*) of non-crossing arcs. We say then that the dependency graph is k-planar, meaning that the minimum number of planes into which A can be distributed is equal to k. The dependency tree displayed in Figure 7c is 2-planar, since the crossing arc $(4 \xrightarrow{\text{case}} 7)$ needs to be located in a second plane.

Projective encodings (4-bit, hexatagging) only recover the full set of arcs when the dependency tree is projective (i.e., 1-planar). Although most of the English treebanks can be covered at $> 99\%$ by projective encodings, Amini et al. (2023) used a pseudo-projectivity transformation[13] to train the hexatagger. For our experiments, we also applied pseudo-projectivity to train the parsers with projective tree encodings.

Hexatagging Amini et al. (2023) encode a dependency tree $G = (W, A)$ with a sequence of labels where each label $\ell_i = (h_i, f_i, r_i)$ has three components: $h_i \in \{\nearrow, \nwarrow\}$, $f_i \in \{\Nwarrow^{\mathrm{R}}, \Nwarrow^{\mathrm{L}}, \Nearrow^{\mathrm{R}}, \Nearrow^{\mathrm{L}}\}$ and $r_i \in \mathcal{R}$; constrained to $\ell_1 = (\nearrow, f_1, r_1)$ and $\ell_n = (\nwarrow, \Omega, r_n)$.

The hexatagger first projectivizes a dependency tree (Nivre and Nilsson, 2005) and then transforms

[12] We use the symbol $\mathcal{R}$ to denote the set of arc labels (dependency types).

[13] The pseudo-projectivity proposed by Nivre and Nilsson (2005) is a lossy transformation with three variants (*head*, *path*, and *head+path*) that encodes the information of the crossing arcs in the arc labels—thus increasing the complexity of this label space $\mathcal{R}$.

(a) Projective dependency tree and the bracketing (**B**) and 4-bit encoding (**B4**).

(b) Dependency tree of Figure 7a transformed into a binary constituent tree and encoded with tetratagging.

(c) 2-planar dependency tree encoded with the bracketing (**B**) and 7-bit encoding (**B7**).

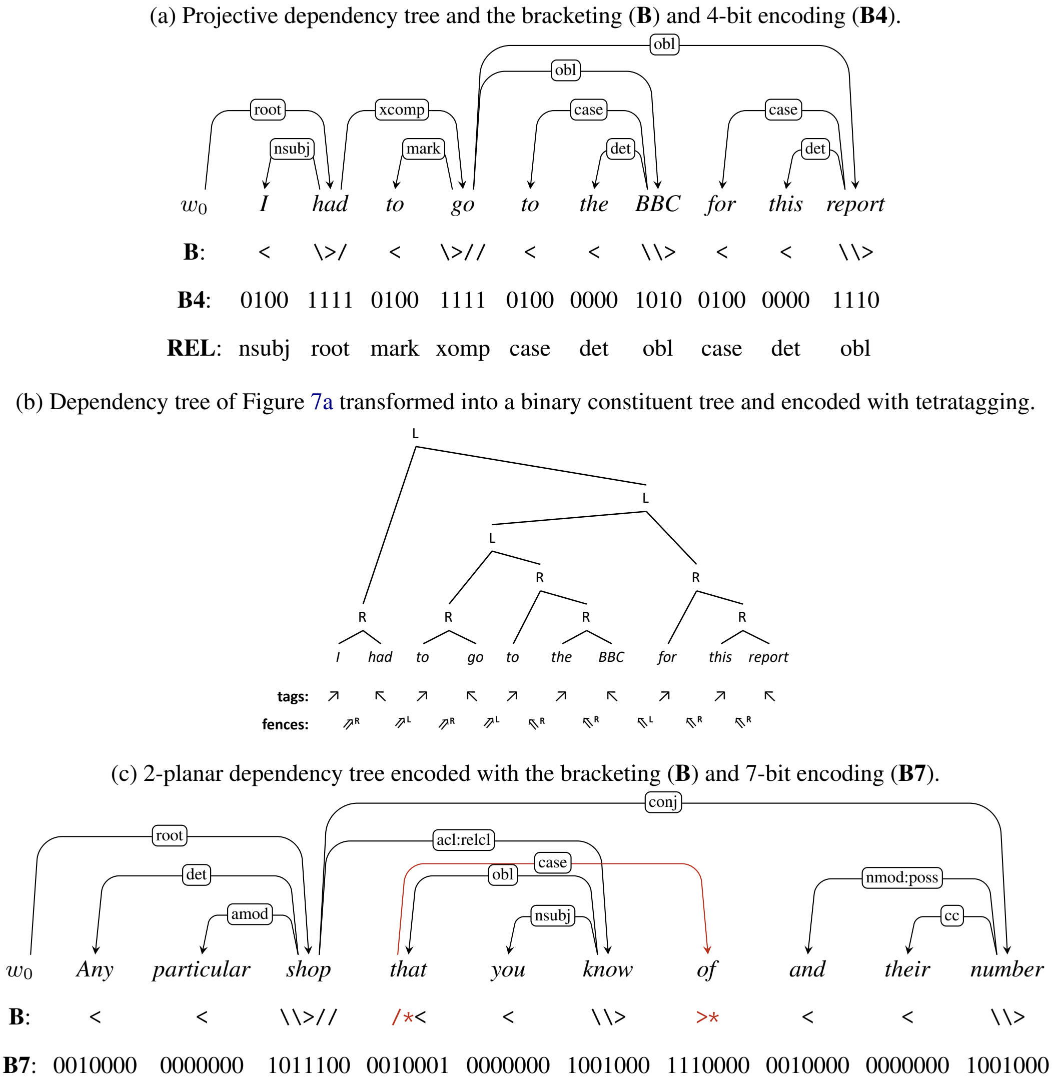

obl
obl
root
xcomp
case
case
nsubj
mark
det
det
w_0 I had to go to the BBC for this report
B: < \>/ < \>// < < \\> < < \\>
B4: 0100 1111 0100 1111 0100 0000 1010 0100 0000 1110
REL: nsubj root mark xomp case det obl case det obl
L
L
L
R
R
R
R
R
R
R
I had to go to the BBC for this report
tags: ↗ ↖ ↗ ↖ ↗ ↗ ↖ ↗ ↗ ↖
fences: ⇗R ⇗L ⇗R ⇗L ⇖R ⇖R ⇖L ⇖R ⇖R
conj
root
acl:relcl
case
det
obl
amod
nsubj
nmod:poss
cc
w_0 Any particular shop that you know of and their number
B: < < \\>// /*< < \\> >* < < \\>
B7: 0010000 0000000 1011100 0010001 0000000 1001000 1110000 0010000 0000000 1001000

Figure 7: Projective (Figures 7a and 7b) 2-planar (Figure 7c) dependency tree examples.

it into a binary constituent tree (BHT) with constituents in $\mathcal{C} = (\mathsf{R}, \mathsf{L})$ that is encoded with tetratagging (Kitaev and Klein, 2020). Figure 7b shows the resulting BHT from the projective dependency tree in Figure 7a. The first component of the label h_i corresponds to the first label of tetratagging, and the second component f_i to the concatenated fencepost symbols and their corresponding constituent. The third component is the incoming arc label of w_i (row **REL** in Figure 7a).

Bracketing encoding Strzyz et al. (2019) use bracket symbols $B = \{/, >, <, \backslash\}$ to encode the information of the incoming and outgoing arcs of each node. Under the bracketing encoding, each label is defined as $\ell_i = (b_i, r_i)$. The first symbol b_i follows the regular expression `\*(>|<)/*` and the presence of each bracket represents an incoming (>, <) or outgoing (/,\) arc from or to a specific direction (left, right) and r_i is the incoming arc label of w_i. When using only the symbols in B, the bracketing encoding only covers sets of arcs with no crossing arcs in the same direction. Strzyz et al. (2019) extended B with $B^* = \{$/*, >*, <*, *$\}$ to only encode arcs of the second plane, thus supporting k-planar dependency trees where $k \leq 2$.

4-bit and 7-bit encodings Gómez-Rodríguez et al. (2023) proposed two bit-based encodings for projective and 2-planar dependency trees. In both variants, each label consists of two components: $\ell_i = (b_i, r_i)$, where $b_i \in \{0, 1\}^m$ is a sequence of m bits ($m = 4$ or $m = 7$, respectively) and r_i is the dependency label of the incoming arc to w_i.

Let $b_i = (b_i^0, b_i^1, b_i^2, b_i^3)$ be the bit symbols of ℓ_i in the 4-bit encoding: b_i^0 is activated if w_i has a left parent (otherwise, it is set to 0); b_i^1 is activated if w_i is the outermost dependent of its parent in the same direction; b_i^2 is activated if w_i has left dependents; and b_i^3 is activated if w_i has right dependents. Figure 7a shows an example of the 4-bit encoding. Note that the label $\ell_5 = (0100, \text{case})$ since the head of w_5 is located to the right ($7 \rightarrow 5$) and w_5 is the leftmost dependent of w_7, and there are no arcs where w_5 is the head.

The 7-bit encoding extends the number of bits of the 4-bit encoding to 7 bits, so $b_i = (b_i^0, b_i^1, b_i^2, b_i^3, b_i^4, b_i^5, b_i^6)$. The first two bits $b_i^0 b_i^1$ encode the plane and position of the head of w_i, so $b_i^0 b_i^1 \in \{00, 01, 10, 11\}$ if w_i has a right or left head in the first plane ($b_i^0 b_i^1 = 00$ or $b_i^0 b_i^1 = 10$, respectively) or second plane ($b_i^0 b_i^1 = 01$ or $b_i^0 b_i^1 = 11$, respectively); b_i^2 is activated if w_i is the outermost dependent of its head in the same direction; b_i^3 and b_i^4 are activated if w_i has left or right dependents in the first plane, respectively; and b_i^5 and b_i^6 are similarly activated for the dependencies of the second plane. See Figure 7c for an example of the 7-bit encoding in a 2-planar tree.

C.3 Graph parsing

Graph parsing relaxes the connectivity, acyclicity, and single-head constraints of a dependency tree. Given a dependency graph $G = (W, A)$, where $A = \{(h \xrightarrow{r} d : d \neq h, r \in \mathcal{R}\}$, we relied on the encodings proposed by Ezquerro et al. (2024b) to represent the arc information as a sequence of n labels, where each label is always defined by two components $\ell_i = (x_i, \rho_i)$, where x_i is configured depending on the encoding algorithm and ρ_i remains constant as the concatenation of incoming arc labels ordered by the absolute position of the heads.

Relative and bracketing encoding The relative and bracketing encodings for dependency trees (Strzyz et al., 2019) can be directly applied to graphs, as displayed in Figure 8. Due to the single-head constraint of dependency trees, under relative encoding each label only contains one head position. For graphs, since each node might have an arbitrary number of heads, the symbol b_i is defined as the sorted sequence of head positions for w_i. The bracketing encoding is independent of tree constraints, as it only encodes the arc information for each individual node.

$4k$ and $6k$-bit encodings The bit-based encodings proposed by Ezquerro et al. (2024b) encode a set of arcs A by (i) first distributing the arcs of A into k mutually-exclusive subsets $\{P_1 \cdots P_k\}$ where each $P_j \subseteq A$ satisfies certain conditions, (ii) then encoding each subset P_j with n symbols $(b_{1,j} \cdots b_{n,j})$ where each symbol $b_{i,j} \in \{0,1\}^{4|6}$ is a sequence of 4 or 6 bits, respectively; and (iii) finally concatenating each symbol at token level to produce $\ell_i = (b_{i,1} \cdots b_{i,k}, r_i)$.

In the $4k$-bit encoding, each subset P_j satisfies that: (i) no arc in P_j crosses another arc in P_j in the same direction, and (ii) there is one and only one incoming arc per node. Then, the bit values of $b_{i,j}$ are assigned as in the 4-bit dependency encoding of Gómez-Rodríguez et al. (2023). Since the second constraint cannot be satisfied if there are nodes without heads, the $4k$-bit encoding creates artificial arcs—connected to the previous node—that are labeled with a null type and discarded in the postprocessing step.

In the $6k$-bit encoding, each subset P_j is instead constrained by (i) not having crossing arcs in the same direction, and (ii) each node having at most one incoming arc per direction. The symbol $b_{i,j} = (b_{i,j}^0 b_{i,j}^1 b_{i,j}^2 b_{i,j}^3 b_{i,j}^4 b_{i,j}^5)$ activates the first (second) bit $b_{i,j}^0$ ($b_{i,j}^1$) with the presence of a left (right) parent of w_i in P_j. The third (fourth) bit $b_{i,j}^2$ ($b_{i,j}^3$) is activated if w_i is the farthest dependent of its left (right) head. The fifth (sixth) bit $b_{i,j}^4$ is activated if w_i has left (right) dependents.

For illustrative examples and details regarding the bit-based graph encodings, we recommend reading Ezquerro et al. (2024b).

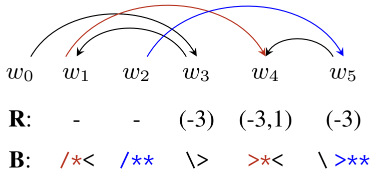


Figure 8: Dependency graph example encoded with the relative (**R**) and bracketing (**B**) encoding with $k = 3$.

D Additional results

In this section we present the detailed performance of our taggers on the five NLP tasks introduced in Section 4 (PoS tagging, NER, dependency parsing, constituency parsing and graph parsing). Tables 7 to 16 break down the constituency and PoS tagging performance of the original PTB (Table 7) and CTB (Table 8) annotations and the SPMRL datasets (Tables 9-16). Tables 17 to 26 show the dependency and PoS tagging performance of the PTB and CTB dependency conversions and the UD treebanks. Tables 27 to 33 show the graph parsing performance on the SDP (Oepen et al., 2015) and IWPT (Bouma et al., 2021) selected datasets.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	62.74	57.14	1.74	1.12	59.98	52.65	3.56	2.69	95.97
$\overset{ϟ}{\rightarrow}$	64.62	59.36	1.66	0.95	61.37	54.16	3.81	2.57	95.74
↔	87.19	84.43	20.74	18.67	85.42	82.14	29.06	25.79	96.27
$\overset{ϟ}{\leftrightarrow}$	90.11	87.94	28.44	25.91	87.7	84.87	34.56	31.33	96.92
DiT	94.47	92.76	40.98	36.3	93.37	90.03	50.91	39.98	95.83
GaT	**95.89**	**94.96**	**50.95**	**47.72**	**94.76**	**93.43**	**57.45**	**53.56**	**97.91**
SSD	68.25	63.16	2.32	1.28	65.08	58.72	4.64	3.23	97.33
Base	95.84	94.88	52.07	49.21	95.19	93.97	59.15	55.5	97.97

Table 7: PTB performance. We refer to the LSTM as (→) and the BiLSTM as (↔). The symbol ϟ indicates that the recurrent cell is replaced by the xLSTM. Same notation as in Table 3.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	68.11	54.79	9.16	7.43	65.47	55.46	9.58	8.85	92.71
$\overset{ϟ}{\rightarrow}$	68.99	55.63	8.64	6.54	66.62	56.33	9.21	8.85	91.95
↔	83.48	76.32	14.45	13.09	81.29	74.91	15.03	13.46	92.74
$\overset{ϟ}{\leftrightarrow}$	86.32	79.64	18.85	16.54	84.13	78.8	19.84	17.75	93.55
DiT	92.81	87.84	30.68	25.03	92.84	87.87	32.36	24.14	97.1
GaT	**94.16**	**90.77**	**36.28**	**32.41**	**93.41**	**93.52**	**39.01**	**35.08**	**97.96**
SSD	70.57	57.35	8.59	6.28	67.81	57.7	8.69	7.96	93.38
Base	94.01	90.84	36.07	32.25	92.81	90.04	36.39	32.36	97.79

Table 8: CTB performance. Notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	39.44	32.78	7.0	5.5	58.31	39.58	8.78	5.54	63.77
$\overset{ϟ}{\rightarrow}$	44.25	37.59	7.72	5.84	57.99	40.36	8.28	5.48	68.84
↔	68.99	63.08	23.02	20.78	79.03	72.22	24.94	22.38	65.82
$\overset{ϟ}{\leftrightarrow}$	75.4	70.68	30.08	27.72	83.13	77.29	32.92	29.64	71.48
DiT	94.2	88.05	60.02	50.1	43.51	36.55	0.0	0.0	65.22
GaT	**94.75**	**91.22**	**62.34**	**58.14**	**94.02**	**91.26**	**59.96**	**56.4**	**79.54**
SSD	47.76	40.49	8.54	6.3	45.22	39.41	7.6	6.1	65.3
Base	91.89	90.22	58.82	56.32	94.12	92.12	60.24	57.14	78.88

Table 9: German-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	65.76	52.21	0.32	0.11	66.56	46.64	0.42	0.11	67.28
$\overset{ϟ}{\rightarrow}$	65.77	50.9	0.74	0.21	65.91	55.81	1.06	0.63	70.26
↔	77.29	66.19	7.19	4.76	78.98	72.58	11.63	8.77	32.72
$\overset{ϟ}{\leftrightarrow}$	79.64	68.78	10.78	5.92	81.33	74.62	15.75	11.73	71.21
DiT	88.07	80.2	19.87	10.25	87.03	79.16	18.67	8.54	54.07
GaT	**92.21**	**89.24**	**37.84**	**31.92**	**89.58**	**85.55**	**25.26**	**19.87**	**72.99**
SSD	63.23	48.55	0.32	0.0	55.81	37.04	0.63	0.32	59.47
Base	92.28	89.17	40.38	32.66	92.4	89.29	47.78	37.74	74.14

Table 10: Basque-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	53.67	43.05	1.34	0.98	16.46	1.67	0.2	0.08	79.03
$\overset{ϟ}{\rightarrow}$	54.12	43.75	1.5	0.87	56.87	46.66	3.66	2.4	80.12
↔	72.6	65.86	10.47	9.45	72.32	66.37	9.88	8.62	79.36
$\overset{ϟ}{\leftrightarrow}$	76.6	70.79	12.48	11.02	77.61	72.63	13.73	12.0	85.55
DiT	88.04	84.91	21.37	18.38	62.75	52.5	2.44	1.3	78.08
GaT	**88.82**	**86.52**	**21.96**	**20.15**	**86.13**	**83.25**	**21.21**	**19.36**	**88.6**
SSD	57.35	47.44	1.73	1.34	52.46	44.23	1.77	1.18	80.05
Base	87.23	83.39	22.98	20.78	87.74	84.79	26.01	24.12	88.55

Table 11: French-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	82.46	71.6	1.68	0.42	83.36	79.53	3.21	1.82	84.43
$\overset{ϟ}{\rightarrow}$	82.02	70.06	1.82	0.42	82.97	78.62	3.35	1.4	81.99
↔	6.88	75.38	3.77	2.09	78.6	78.0	0.7	0.42	84.54
$\overset{ϟ}{\leftrightarrow}$	87.54	75.91	6.15	3.35	88.39	85.25	9.92	7.96	84.01
DiT	94.54	91.17	18.44	10.47	90.42	82.78	11.45	3.77	90.98
GaT	**95.15**	**92.47**	**21.51**	**16.34**	**94.02**	**91.1**	**26.4**	**19.83**	**95.36**
SSD	79.22	61.61	1.12	0.14	77.72	58.83	0.7	0.28	69.11
Base	95.09	92.43	19.55	14.39	93.85	90.82	24.86	18.3	95.36

Table 12: Hebrew-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	73.66	60.32	4.26	2.18	74.09	60.19	3.96	1.78	15.34
$\overset{ϟ}{\rightarrow}$	71.24	57.42	2.87	1.09	72.73	59.06	2.28	0.89	**75.63**
↔	9.65	8.78	0.1	0.0	79.89	70.23	11.6	7.23	58.15
$\overset{ϟ}{\leftrightarrow}$	80.06	70.7	11.2	7.93	80.25	71.83	12.78	8.92	71.05
DiT	94.41	89.4	45.79	33.0	90.14	79.83	15.98	3.71	61.17
GaT	**95.11**	**92.42**	**51.64**	**43.11**	**93.29**	**90.45**	**35.68**	**29.93**	74.28
SSD	70.8	55.78	2.38	0.99	65.5	49.56	1.39	0.1	60.05
Base	95.1	92.47	52.82	43.11	93.28	90.97	39.84	33.99	75.71

Table 13: Hungarian-SPMRL performance. Same notation as in Table 7.

	R				**H**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	60.69	52.52	0.83	0.31	66.14	54.81	4.5	1.57	60.51
$\overset{ϟ}{\rightarrow}$	58.93	50.47	0.7	0.26	66.75	56.32	4.81	2.01	65.24
↔	67.23	60.2	8.48	5.55	74.68	67.72	12.42	8.92	49.74
$\overset{ϟ}{\leftrightarrow}$	68.06	61.61	9.49	6.95	76.66	71.0	16.0	12.51	65.52
DiT	89.29	85.18	34.89	27.2	85.66	79.16	29.09	17.91	46.65
GaT	**89.95**	**87.6**	**40.93**	**35.33**	**87.15**	**84.01**	**31.31**	**25.27**	**66.56**
SSD	52.49	41.18	0.0	0.0	47.74	31.46	0.13	0.0	33.29
Base	89.43	86.92	38.61	33.36	89.77	86.93	40.27	33.54	66.42

Table 14: Korean-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
→	80.46	63.39	3.28	1.09	77.85	64.95	5.11	1.82	61.76
$\overset{ϟ}{\rightarrow}$	79.75	59.61	3.89	1.7	82.4	73.19	9.49	4.74	64.7
↔	72.62	30.46	0.0	0.0	91.24	87.65	32.73	28.1	42.71
$\overset{ϟ}{\leftrightarrow}$	88.58	74.6	26.4	14.48	92.39	89.47	38.32	34.06	64.34
DiT	**97.04**	93.43	59.61	48.42	**97.03**	**94.07**	**62.9**	**48.3**	48.55
GaT	96.9	**94.39**	**59.85**	**49.15**	96.28	93.6	48.66	37.1	**66.1**
SSD	79.5	62.77	1.09	0.36	75.2	53.23	2.55	1.22	53.52
Base	96.9	94.46	60.95	51.46	97.49	95.85	69.46	62.04	66.92

Table 15: Polish-SPMRL performance. Same notation as in Table 7.

	R				**T**				**POS**
	UF	LF	UM	LM	UF	LF	UM	LM	
$\rightarrow$	55.06	48.81	5.11	3.9	59.28	46.26	6.01	4.35	80.51
$\overset{\lightning}{\rightarrow}$	54.23	47.43	3.75	2.7	59.33	46.31	6.46	4.65	76.46
$\leftrightarrow$	13.7	12.52	0.15	0.0	24.87	14.94	3.75	0.15	78.39
$\overset{\lightning}{\leftrightarrow}$	70.88	63.03	12.76	6.01	73.46	62.72	13.21	10.36	80.08
DiT	90.46	81.26	34.08	21.92	84.59	66.95	19.84	7.49	89.82
GaT	**90.77**	**88.02**	**39.19**	**34.53**	**88.31**	**84.53**	**42.19**	**37.84**	**94.38**
SSD	51.51	44.68	3.3	2.55	51.56	40.57	3.45	1.95	77.36
Base	90.67	88.12	41.14	36.49	89.69	86.07	45.35	41.29	94.47

Table 16: Swedish-SPMRL performance. Same notation as in Table 7.

	B				B4				B7				H			
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM
→	65.97	60.01	6.17	3.56	66.28	60.42	6.0	3.1	65.99	60.05	5.46	2.86	58.86	53.96	5.75	3.02
↯→	67.64	61.53	6.66	3.68	68.35	62.63	6.33	2.86	68.28	62.65	6.58	3.52	61.71	57.05	5.75	2.94
↔	90.16	88.51	35.97	30.67	89.51	87.92	33.94	29.01	90.05	88.51	36.34	31.37	88.22	86.35	32.49	26.82
↯↔	91.92	90.41	41.35	35.68	91.51	90.03	40.73	34.98	91.34	90.01	39.94	35.43	91.45	90.14	42.09	36.67
DiT	89.39	86.83	35.02	26.2	94.4	92.31	55.05	40.31	94.79	92.39	57.66	41.39	94.44	92.04	55.26	38.37
GaT	**95.77**	**94.13**	**61.96**	**49.05**	**95.87**	**94.19**	**63.37**	**49.92**	**95.85**	**94.19**	**63.87**	**50.37**	**95.78**	**94.13**	**63.78**	**49.75**
SSD	72.45	66.25	7.91	3.81	71.45	65.52	6.95	3.56	72.05	66.07	6.58	3.1	64.84	59.87	6.21	2.86
Base	95.57	93.86	59.35	46.36	95.96	94.34	63.2	49.63	95.78	94.05	62.83	48.26	95.76	94.02	62.58	49.13

Table 17: Performance in the PTB with dependency annotations. Same notation as in Tables 1, 4 and 7.

	B				B4				B7				H			
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM
→	57.75	45.13	9.97	7.19	58.13	45.42	10.71	7.19	58.13	45.42	10.71	7.19	54.96	42.52	11.29	6.88
↯→	61.81	50.41	12.44	8.87	61.12	49.5	12.86	9.08	61.12	49.5	12.86	9.08	56.65	45.36	12.34	8.82
↔	82.12	79.3	25.93	21.21	81.9	79.3	26.93	22.52	81.9	79.3	26.93	22.52	79.42	76.31	25.14	20.47
↯↔	84.16	82.18	30.08	25.3	83.19	80.82	27.82	23.57	83.19	80.82	27.82	23.57	82.15	79.68	29.29	24.36
DiT	84.29	82.21	28.29	24.51	90.17	87.93	42.2	33.91	89.63	87.42	40.79	33.23	89.28	86.65	41.78	32.34
GaT	**90.86**	**88.84**	**42.05**	**34.44**	**91.17**	**89.19**	**46.14**	**37.59**	**90.27**	**88.14**	**43.25**	**35.17**	**90.83**	**88.57**	**45.83**	**36.17**
SSD	57.91	44.74	10.08	7.51	57.66	44.47	9.71	6.61	57.84	44.78	10.29	7.09	54.05	41.12	10.03	6.46
Base	90.42	88.28	41.26	33.44	90.8	88.63	44.25	35.43	90.58	88.47	44.3	35.17	90.32	87.94	44.83	34.96

Table 18: Performance in the CTB with dependency annotations. Same notation as in Table 17.

	B				B4				B7				H				U	X
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM		
→	60.99	51.89	7.57	2.46	61.49	53.4	7.37	2.76	60.94	52.63	6.96	2.46	55.91	48.96	8.5	3.48	90.31	89.99
↯→	69.43	60.81	11.87	4.81	64.12	56.59	9.62	3.17	63.9	56.09	11.05	5.12	58.27	51.98	8.5	3.68	88.96	88.31
↔	77.22	67.42	18.83	7.27	76.44	67.25	19.86	9.01	76.82	66.91	20.06	8.19	75.63	67.21	21.8	9.72	91.52	91.79
↯↔	81.87	74.22	25.18	13.2	81.68	73.87	27.43	14.02	82.9	76.07	28.56	15.97	81.95	75.8	29.99	16.99	91.36	92.26
DiT	75.74	70.28	20.78	11.77	**86.44**	80.75	**38.38**	19.45	85.38	79.74	37.67	20.06	85.14	79.22	35.62	16.89	**96.62**	94.62
GaT	**87.67**	**82.65**	**39.3**	**22.82**	86.36	**81.67**	38.18	**22.52**	**87.92**	**83.14**	**42.99**	**24.56**	**87.55**	**82.68**	**42.78**	**23.34**	96.21	**97.42**
SSD	61.71	52.34	7.78	2.56	59.6	50.4	6.55	1.84	59.06	50.53	6.55	2.05	55.35	47.86	7.98	3.48	93.16	92.67
Base	87.33	82.49	38.38	22.11	86.2	81.13	39.0	22.11	87.44	82.77	41.15	23.23	87.28	82.29	42.58	22.93	96.79	97.53

Table 19: Performance in the German-GSD. Same notation as in Table 17.

	B				B4				B7				H				U
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	
→	52.02	41.85	4.56	2.06	54.21	44.77	4.84	2.11	54.75	44.51	5.34	2.22	44.97	36.92	4.67	1.5	87.65
↯→	55.7	45.48	5.89	2.22	53.1	43.31	5.11	2.06	53.72	43.86	5.28	1.78	44.53	36.63	5.23	2.17	85.06
↔	70.03	57.46	12.95	4.5	70.43	58.34	15.4	5.06	69.11	56.36	14.01	4.17	67.93	56.76	17.01	5.84	81.76
↯↔	78.18	68.11	22.68	8.78	77.05	67.9	22.57	10.56	78.0	68.44	22.9	9.34	75.91	66.8	24.18	9.73	85.56
DiT	59.0	53.77	4.17	2.33	84.69	79.43	42.3	24.51	83.63	78.86	40.08	25.57	85.25	78.47	41.47	19.46	93.54
GaT	**86.0**	**81.76**	**42.47**	**28.74**	**86.2**	**81.93**	**45.41**	**31.13**	**86.64**	**82.54**	**46.47**	**31.63**	**86.06**	**81.76**	**44.91**	**29.41**	**95.54**
SSD	41.57	28.92	1.56	0.56	43.26	31.34	2.5	0.72	42.83	30.41	2.22	0.5	37.66	26.66	2.89	0.78	88.12
Base	86.6	82.43	43.58	29.85	86.12	81.89	44.36	30.24	86.89	82.58	46.75	31.02	86.64	82.4	46.8	31.57	95.52

Table 20: Performance in the Basque-BDT. Same notation as in Table 17.

	B				B4				B7				H				U
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	
$\rightarrow$	73.06	65.31	4.81	1.44	73.63	66.27	4.81	2.16	73.61	66.07	3.61	1.2	66.27	60.32	4.33	1.44	94.81
$\overset{\lightning}{\rightarrow}$	76.22	69.62	5.77	2.88	75.34	68.39	5.05	1.68	74.99	68.35	7.45	3.85	67.99	62.0	5.53	1.92	94.04
$\leftrightarrow$	86.59	80.85	20.43	10.1	86.36	81.8	19.47	12.74	84.47	78.14	18.27	7.45	85.02	79.57	21.39	10.1	94.37
$\overset{\lightning}{\leftrightarrow}$	88.44	84.61	22.84	15.62	88.27	84.29	26.44	16.35	88.82	84.75	24.04	13.22	89.28	84.9	29.57	16.59	95.62
DiT	88.95	85.94	32.93	23.08	93.47	90.57	46.39	**32.21**	93.12	90.34	43.03	30.05	94.32	90.78	49.04	29.81	98.09
GaT	**94.64**	**91.99**	**47.84**	**33.89**	**93.99**	**91.05**	**49.04**	32.21	**94.47**	**91.94**	**50.0**	**36.78**	**94.84**	**91.97**	**50.0**	**35.82**	**98.42**
SSD	73.44	66.59	5.05	2.16	74.44	67.09	6.73	2.4	74.44	67.39	5.77	2.64	66.14	59.99	4.09	1.68	96.21
Base	94.19	91.42	46.15	32.93	93.91	91.1	50.24	35.1	94.27	91.59	46.63	32.93	94.53	91.5	50.48	33.41	98.45

Table 21: Performance in the French-GSD. Same notation as in Table 17.

	B				B4				B7				H				U
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	
$\rightarrow$	68.43	63.89	2.65	1.43	68.67	64.25	1.83	0.81	69.49	64.59	1.43	1.22	59.31	55.73	0.81	0.61	92.47
$\overset{\lightning}{\rightarrow}$	70.95	66.75	2.24	1.02	69.77	65.84	1.63	1.02	70.11	65.71	2.04	0.81	62.16	58.56	1.43	0.81	90.3
$\leftrightarrow$	77.55	71.56	8.35	4.07	80.04	74.02	11.61	5.7	78.76	72.38	9.16	4.07	74.87	68.54	11.0	4.28	91.41
$\overset{\lightning}{\leftrightarrow}$	82.07	78.25	13.44	7.74	83.35	79.3	16.29	9.37	84.11	80.31	18.13	12.02	81.69	77.58	15.07	9.57	92.05
DiT	81.58	77.97	12.83	8.15	89.67	85.3	30.55	17.52	90.42	86.43	32.59	19.55	90.98	85.87	34.62	15.68	95.24
GaT	**91.75**	**88.43**	**38.09**	**24.85**	**92.5**	**89.55**	**39.92**	**27.29**	**91.33**	**88.18**	**36.46**	**23.63**	**92.19**	**88.98**	**40.33**	**25.87**	**97.88**
SSD	57.47	47.68	1.22	0.41	60.17	49.03	0.41	0.2	60.67	49.97	0.41	0.0	50.01	41.78	0.61	0.41	85.34
Base	91.99	88.75	37.88	25.87	91.67	88.37	39.1	25.25	92.24	89.12	38.49	25.46	92.39	89.26	41.96	27.7	97.52

Table 22: Performance in the Hebrew-HTB. Same notation as in Table 17.

	B				B4				B7				H				U
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	
$\rightarrow$	51.77	41.07	0.89	0.22	50.41	40.19	1.56	0.22	51.37	39.72	0.67	0.0	41.31	31.78	1.11	0.0	77.75
$\overset{\lightning}{\rightarrow}$	56.46	43.54	1.78	0.0	53.97	42.12	0.45	0.0	52.1	41.45	0.89	0.22	44.65	35.79	1.78	0.0	74.88
$\leftrightarrow$	25.09	3.92	0.0	0.0	61.85	43.19	3.79	0.67	34.82	10.31	0.89	0.0	33.06	5.54	0.0	0.0	75.71
$\overset{\lightning}{\leftrightarrow}$	67.64	54.46	4.68	1.34	66.38	55.05	4.9	0.89	65.63	52.85	3.79	0.89	64.48	52.56	6.24	1.11	67.06
DiT	60.87	54.76	1.34	0.22	79.27	72.16	15.14	6.46	71.63	64.94	8.91	4.9	28.96	21.62	1.34	0.45	95.08
GaT	**83.44**	**77.67**	**16.26**	**8.02**	**85.26**	**79.51**	**24.5**	**12.25**	**85.19**	**79.29**	**23.16**	**13.14**	**83.59**	**77.58**	**23.16**	**11.14**	**95.81**
SSD	35.81	22.11	0.22	0.22	37.01	23.19	0.45	0.22	37.46	23.09	0.0	0.0	33.25	21.47	0.89	0.22	76.42
Base	83.28	77.43	15.81	8.24	85.64	79.7	24.05	11.58	85.71	80.12	24.5	13.36	85.17	79.23	25.61	12.47	96.28

Table 23: Performance in the Hungarian-Szeged. Same notation as in Table 17.

	B				B4				B7				H				U	X
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM		
$\rightarrow$	63.35	52.31	6.73	2.58	64.72	53.05	5.86	1.88	63.88	52.85	6.34	2.27	65.59	52.5	11.19	3.1	79.89	66.57
$\overset{\lightning}{\rightarrow}$	67.38	55.04	9.58	2.93	65.67	54.25	6.52	2.19	65.64	54.88	6.34	3.02	63.05	50.93	9.14	2.93	78.64	65.13
$\leftrightarrow$	76.22	67.21	15.17	7.26	75.89	66.83	16.18	7.91	75.21	66.1	15.7	7.3	75.95	67.71	17.88	9.93	80.33	64.92
$\overset{\lightning}{\leftrightarrow}$	80.19	72.05	21.86	11.28	80.22	72.85	21.73	12.33	79.3	71.67	21.16	12.33	78.52	71.01	20.64	12.2	80.54	67.1
DiT	84.68	79.75	31.18	20.33	85.6	79.67	33.19	17.93	85.71	80.87	34.59	22.52	87.25	81.52	38.39	22.96	91.14	81.4
GaT	**88.65**	**84.28**	**42.15**	**29.12**	**87.77**	**83.43**	**40.97**	**28.25**	**88.04**	**83.79**	**41.1**	**28.82**	**88.09**	**83.59**	**42.15**	**29.21**	**94.67**	**85.33**
SSD	46.48	21.24	0.79	0.04	47.57	22.4	0.39	0.0	47.58	22.52	0.35	0.0	45.53	21.8	1.31	0.04	65.05	41.17
Base	88.19	83.63	40.53	27.11	87.5	83.09	39.88	27.42	87.68	83.12	41.15	26.98	87.49	83.0	41.01	27.94	94.49	85.67

Table 24: Performance in the Korean-KAIST. Same notation as in Table 17.

	B				**B4**				**B7**				**H**				U	**X**
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM		
$\rightarrow$	68.26	59.84	10.56	5.91	67.33	59.39	9.26	5.64	66.77	58.62	8.85	5.1	56.17	50.04	8.17	4.7	90.35	76.23
$\overset{\lightning}{\rightarrow}$	72.4	64.15	11.38	7.54	69.72	61.34	10.11	6.0	69.53	61.41	9.89	6.23	60.88	54.99	11.11	7.27	89.33	75.44
$\leftrightarrow$	83.69	73.52	29.71	14.99	82.71	72.25	30.11	14.76	82.63	71.81	30.43	14.81	82.22	72.48	31.06	15.62	89.32	77.06
$\overset{\lightning}{\leftrightarrow}$	88.73	80.4	41.17	21.99	89.12	81.04	42.57	23.43	88.97	80.64	42.08	22.26	88.51	80.3	43.21	22.71	91.09	79.47
DiT	90.2	85.45	50.74	32.37	94.51	89.06	63.61	35.67	94.42	89.3	62.8	37.43	94.36	87.56	62.75	30.29	97.53	92.58
GaT	**95.4**	**91.01**	**64.97**	**40.77**	**94.78**	**90.38**	**64.6**	**40.5**	**95.39**	**90.96**	**66.32**	**41.4**	**95.07**	**90.69**	**66.86**	**41.31**	**98.82**	**96.13**
SSD	62.36	52.98	7.77	4.47	61.84	51.86	7.81	3.7	62.52	52.62	6.77	3.61	53.55	45.43	6.77	3.66	90.84	71.47
Base	95.27	90.67	63.97	39.86	94.75	90.36	64.65	40.18	95.39	90.76	66.32	40.99	95.1	90.6	66.59	40.45	98.86	95.54

Table 25: Performance in the Polish-PDB. Same notation as in Table 17.

	B				**B4**				**B7**				**H**				U	**X**
	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM	UAS	LAS	UM	LM		
$\rightarrow$	64.75	54.52	9.52	2.63	67.85	57.87	11.89	4.18	66.94	56.96	11.24	4.1	59.21	52.49	8.29	4.27	88.74	83.39
$\overset{\lightning}{\rightarrow}$	69.26	59.46	9.93	3.45	69.88	60.32	10.58	3.69	69.99	59.95	10.66	3.61	62.8	56.48	9.35	4.1	84.87	79.29
$\leftrightarrow$	74.69	65.8	13.62	7.22	77.38	69.74	17.64	10.5	76.44	67.34	16.98	8.94	72.93	64.06	19.52	7.88	84.5	80.17
$\overset{\lightning}{\leftrightarrow}$	82.93	76.48	22.72	14.27	84.38	78.06	27.97	17.06	84.09	77.3	25.76	16.0	82.38	76.12	28.71	16.9	90.8	82.32
DiT	74.22	72.02	17.15	14.68	92.47	89.54	53.24	41.18	91.3	88.52	49.38	38.23	90.49	84.5	44.63	23.46	97.67	92.14
GaT	**93.08**	**90.55**	**53.24**	**42.74**	**93.34**	**90.87**	**56.77**	**45.04**	**93.91**	**91.6**	**58.57**	**47.83**	**93.2**	**90.42**	**56.19**	**43.4**	**98.18**	**95.89**
SSD	48.17	35.66	4.68	0.82	52.32	38.59	5.91	1.07	53.8	40.06	5.99	1.39	46.04	36.79	6.32	2.95	87.59	78.58
Base	93.19	90.86	53.16	42.74	93.58	91.03	56.93	45.53	93.48	91.2	56.6	47.5	92.61	90.04	54.8	43.23	98.22	95.87

Table 26: Performance in the Swedish-Talbanken. Same notation as in Table 17.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	67.2	59.67	0.99	0.92	64.91	55.5	3.4	2.41	56.53	43.34	1.35	0.92	63.11	54.79	3.26	2.2
$\overset{\lightning}{\rightarrow}$	70.81	65.3	1.56	1.13	67.55	60.52	3.97	2.98	59.01	45.98	1.28	1.06	66.25	59.24	3.48	2.55
$\leftrightarrow$	85.81	80.5	17.3	13.69	87.26	82.81	24.04	19.65	86.38	82.2	20.0	17.09	87.07	83.02	22.48	18.72
$\overset{\lightning}{\leftrightarrow}$	88.57	84.53	23.62	19.22	90.31	86.7	33.19	26.88	89.38	85.77	27.16	23.19	89.36	85.58	29.36	23.26
DiT	79.9	75.05	16.81	14.54	93.16	85.91	50.28	36.03	90.84	84.73	37.87	31.84	92.6	85.73	48.65	36.38
GaT	**91.74**	**89.77**	**35.53**	**32.2**	**95.31**	**93.57**	**56.6**	**49.57**	**94.57**	**92.82**	**48.94**	**43.83**	**95.07**	**93.42**	**55.82**	**49.5**
SSD	69.51	64.37	1.28	0.99	71.79	63.66	4.68	2.77	62.45	48.33	1.56	1.06	69.43	61.44	4.11	2.91
Base	91.8	89.77	35.39	31.77	95.31	93.48	56.45	48.79	94.38	92.46	47.59	42.27	95.11	93.37	53.55	46.17

Table 27: Performance in the English-DM dataset. Same abbreviations as in Table 5: relative (**R**), bracketing (**B**), $4k$-bit ($\boldsymbol{4k}$) and $6k$-bit ($\boldsymbol{6k}$) encodings.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	71.31	63.64	4.56	3.53	67.05	60.02	4.26	3.24	70.43	60.64	3.53	2.35	68.89	61.7	4.12	2.94
$\overset{\lightning}{\rightarrow}$	75.02	67.86	5.59	3.53	69.9	63.17	5.29	4.85	71.57	63.14	5.29	4.26	69.45	62.62	4.85	3.68
$\leftrightarrow$	76.92	68.14	7.21	5.44	77.33	68.51	7.79	6.03	75.95	67.45	7.79	5.88	75.76	67.07	7.79	5.44
$\overset{\lightning}{\leftrightarrow}$	80.24	72.53	9.41	6.91	80.77	73.15	10.74	6.47	80.49	72.85	10.15	6.76	80.35	73.31	8.97	6.47
DiT	78.79	72.12	8.09	5.0	77.88	69.52	8.97	5.59	82.91	73.37	9.85	5.44	85.31	76.61	15.0	7.79
GaT	**84.85**	**78.24**	**11.91**	**8.53**	**87.66**	**80.39**	**17.5**	**8.68**	**88.07**	**81.3**	**17.79**	**10.29**	**88.52**	**81.75**	**18.97**	**10.29**
SSD	68.44	57.09	4.41	2.94	62.82	52.67	3.68	2.35	67.14	53.31	3.97	3.24	64.92	52.95	4.26	2.94
Base	84.96	78.19	11.91	7.94	87.88	81.08	17.65	9.56	88.05	80.77	19.12	9.12	87.67	80.7	17.94	8.82

Table 28: Graph parsing performance in the Arabic-PADT (IWPT) dataset. Same notation as in Table 27.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	58.36	50.5	4.03	2.06	68.51	60.46	7.35	3.67	70.72	62.28	8.51	4.66	70.38	61.84	7.53	4.12
$\overset{\text{↯}}{\rightarrow}$	65.33	60.0	5.91	4.57	74.06	67.5	11.47	7.62	74.85	67.2	9.77	5.82	74.54	67.78	11.83	7.17
$\leftrightarrow$	83.5	73.55	25.45	11.47	81.41	72.37	21.33	9.77	78.92	69.29	19.98	9.77	82.24	72.94	25.27	11.74
$\overset{\text{↯}}{\leftrightarrow}$	88.06	81.94	33.96	20.97	88.21	81.65	34.5	20.79	88.51	82.6	38.26	23.48	88.99	82.78	38.71	22.85
DiT	84.75	80.65	25.72	17.56	86.15	81.2	35.22	23.57	91.37	85.56	47.58	28.41	92.79	86.83	56.99	32.26
GaT	**91.84**	**87.81**	**47.31**	**32.17**	**94.46**	**90.34**	**59.32**	**39.78**	**94.86**	**90.47**	**62.46**	**39.52**	**95.04**	**90.43**	**64.34**	**40.86**
SSD	50.59	42.54	1.79	1.08	58.6	49.25	5.65	3.23	60.24	50.17	4.57	3.14	60.6	50.31	4.66	3.14
Base	91.75	87.73	46.42	31.45	94.28	90.01	58.87	39.52	94.87	90.64	63.35	41.58	94.91	90.47	64.52	40.5

Table 29: Graph parsing performance in the Bulgarian-BTB (IWPT) dataset. Same notation as in Table 27.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	67.16	60.9	14.25	11.84	73.39	66.31	14.69	13.16	77.13	69.3	15.79	13.82	76.75	69.13	14.69	12.5
$\overset{\text{↯}}{\rightarrow}$	69.89	64.98	13.82	12.5	77.03	70.91	16.67	14.91	78.19	70.58	16.01	14.69	77.62	70.95	16.67	14.47
$\leftrightarrow$	79.88	72.09	19.96	16.67	80.99	74.33	21.49	18.42	81.71	74.19	24.56	18.86	0.26	0.0	0.0	0.0
$\overset{\text{↯}}{\leftrightarrow}$	84.91	80.24	23.46	21.05	84.37	79.88	21.49	18.42	85.95	81.25	26.32	23.46	86.66	82.21	28.07	24.56
DiT	77.62	75.55	16.01	15.13	69.1	65.97	13.38	12.72	86.52	81.72	34.87	25.22	87.35	83.29	37.06	28.29
GaT	**86.81**	**84.49**	**26.75**	**23.9**	**93.16**	**90.39**	**42.76**	**35.31**	**93.65**	**90.84**	**49.78**	**40.13**	**93.92**	**91.73**	**49.78**	**42.32**
SSD	62.33	57.52	13.16	12.5	11.62	0.99	2.85	0.0	72.97	64.07	15.13	12.06	74.63	66.0	13.16	11.62
Base	87.08	84.83	27.85	24.78	92.81	89.94	41.23	33.99	93.3	90.39	47.59	37.28	93.49	90.97	48.25	39.69

Table 30: Graph parsing performance in the French-Sequoia (IWPT) dataset. Same notation as in Table 27.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	66.75	60.39	5.19	3.11	75.99	68.58	9.13	5.19	76.82	68.23	5.6	3.32	76.31	69.11	9.54	6.43
$\overset{\text{↯}}{\rightarrow}$	70.99	66.04	8.51	4.98	77.53	71.37	10.79	6.64	78.87	71.48	11.2	8.71	78.84	72.83	12.24	7.88
$\leftrightarrow$	87.33	81.65	29.25	17.84	86.9	81.49	29.46	19.92	85.9	79.92	28.42	18.46	86.28	80.08	29.46	16.8
$\overset{\text{↯}}{\leftrightarrow}$	89.61	85.53	35.48	26.76	90.18	86.16	37.34	28.63	90.22	86.27	37.97	29.25	90.59	86.69	39.83	29.88
DiT	80.72	77.7	9.34	6.02	85.76	82.61	31.95	25.31	91.11	86.7	41.29	29.88	92.69	87.85	48.96	31.33
GaT	**91.99**	**89.52**	**43.15**	**34.23**	**94.0**	**91.23**	**50.41**	**39.42**	**94.17**	**91.46**	**56.43**	**43.15**	**94.19**	**91.33**	**54.98**	**42.53**
SSD	66.87	61.54	4.98	3.11	75.08	67.93	9.54	5.6	76.27	68.29	8.09	4.77	76.46	69.46	9.75	5.81
Base	91.65	89.2	41.49	32.99	94.08	91.09	52.07	38.17	94.23	91.56	54.98	42.12	94.58	91.87	55.6	42.12

Table 31: Graph parsing performance in the Italian-ISDT (IWPT) dataset. Same notation as in Table 27.

	R				**B**				$\boldsymbol{4k}$				$\boldsymbol{6k}$			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	49.11	40.97	1.17	0.17	59.27	50.18	3.02	0.67	61.99	50.9	2.85	0.67	63.1	53.47	2.35	0.84
$\overset{\text{↯}}{\rightarrow}$	54.77	47.67	1.17	0.67	65.79	58.64	3.52	1.68	66.01	57.26	2.18	0.67	66.56	59.3	4.03	1.85
$\leftrightarrow$	79.69	69.01	15.94	6.71	2.43	0.0	0.0	0.0	71.69	62.38	11.58	5.54	74.03	63.54	11.41	5.2
$\overset{\text{↯}}{\leftrightarrow}$	83.47	77.19	20.97	13.59	83.29	76.5	20.13	13.26	82.29	75.51	23.49	13.59	83.31	76.64	22.65	14.6
DiT	76.72	72.07	8.22	5.03	75.04	69.88	13.42	7.89	85.06	78.46	31.71	19.46	88.34	81.7	44.46	22.82
GaT	**88.52**	**84.1**	**32.89**	**22.15**	**92.11**	**87.14**	**48.32**	**30.37**	**91.56**	**87.03**	**52.52**	**33.05**	**92.18**	**87.75**	**52.52**	**34.9**
SSD	45.92	38.79	0.17	0.0	55.25	47.23	1.34	0.84	57.32	47.91	1.68	1.01	58.05	48.74	2.52	1.34
Base	88.39	83.81	31.21	21.48	92.17	87.07	47.99	29.87	91.75	87.23	50.67	33.05	92.26	87.88	54.36	34.23

Table 32: Graph parsing performance in the Dutch-Alpino (IWPT) dataset. Same notation as in Table 27.

	R				**B**				**$4k$**				**$6k$**			
	UF	LF	UM	LM	UF	LF	UM	LM	UF	LF	UF	LF	UF	LF	UM	LM
$\rightarrow$	42.26	14.6	0.0	0.0	28.78	5.54	0.0	0.0	29.3	4.39	0.0	0.0	38.1	6.88	0.0	0.0
$\overset{\lightning}{\rightarrow}$	55.25	40.02	0.0	0.0	51.9	34.11	0.0	0.0	58.18	37.85	0.83	0.0	57.91	40.49	0.83	0.0
$\leftrightarrow$	38.64	8.53	0.0	0.0	7.37	7.37	0.0	0.0	35.17	6.59	0.0	0.0	36.06	1.75	0.0	0.0
$\overset{\lightning}{\leftrightarrow}$	63.23	48.91	**0.83**	**0.83**	65.48	47.64	2.5	0.0	68.24	52.21	3.33	**0.83**	67.0	51.61	1.67	0.83
DiT	25.94	2.95	0.0	0.0	42.2	12.44	0.0	0.0	20.96	3.97	0.0	0.0	19.4	3.11	0.0	0.0
GaT	**66.45**	**55.61**	0.83	0.0	**74.03**	**60.52**	**6.67**	**0.83**	**76.54**	**62.32**	**8.33**	0.83	**76.29**	**62.79**	**12.5**	**2.5**
SSD	42.32	15.81	0.0	0.0	22.66	5.56	0.0	0.0	33.85	11.46	0.0	0.0	30.23	9.88	0.0	0.0
Base	66.13	54.02	2.5	0.0	74.77	60.56	6.67	0.0	77.06	62.71	12.5	1.67	77.21	63.99	12.5	1.67

Table 33: Graph parsing performance in the Tamil-TTB (IWPT) dataset. Same notation as in Table 27.