From Partial to Strictly Incremental Constituent Parsing

Ana Ezquerro, Carlos Gómez-Rodríguez and David Vilares
Universidade da Coruña, CITIC
Departamento de Ciencias de la Computación y Tecnologías de la Información
Campus de Elviña s/n, 15071
A Coruña, Spain
{ana.ezquerro, carlos.gomez, david.vilares}@udc.es

Abstract

We study incremental constituent parsers to assess their capacity to output trees based on prefix representations alone. Guided by strictly left-to-right generative language models and tree-decoding modules, we build parsers that adhere to a strong definition of incrementality across languages. This builds upon work that asserted incrementality, but that mostly only enforced it on either the encoder or the decoder. Finally, we conduct an analysis against non-incremental and partially incremental models.

1 Introduction

Incremental NLP aims to learn and adapt partial representations as information unfolds. However, with the rise of bidirectional LSTMs (Hochreiter and Schmidhuber, 1997) and Transformers (Vaswani et al., 2017), recent research has focused on non-incremental solutions. These models process the full input for contextualization before they start generating any output. Therefore, this approach does not capture the progressive unfolding of input over time, giving the sense that all of it is available all of a sudden (Madureira and Schlangen, 2020). This is not an issue for most NLP tasks, but it is relevant for others, such as real-time NLP, e.g., instant machine translation or real-time speech. Furthermore, work on incremental processing holds relevance in interdisciplinary research, especially where computer science, linguistics, and cognitive studies intersect.

While some studies have addressed the challenge of outputting incremental structured representations - for various definitions of incrementality (Konstas et al., 2014; Köhn, 2018; Shen et al., 2021) - analyses of trees remain limited, more notably since the popularization of deep learning, and are mostly partially incremental approaches.

In this context, Titov and Henderson (2007), one of the first neural parsing models, was also an incremental network based on sigmoid belief networks. This generative model broke down the probability of a structure into probabilities for individual derivation decisions, each influenced by previous decision history. However, the computation was expensive and its evaluation was restricted to sentences of up to 15 tokens in the English Penn Treebank (Marcus et al., 1993). For shift-reduce constituent parsing, Cross and Huang (2016) proposed an incremental model with minimal features, focusing on only three sentence positions to predict the next action. However, input sentences were contextualized using bidirectional LSTMs, thus relying on non-incremental encoders and effectively considering all upcoming words; a strategy that was later widely adopted by most neural syntactic parsing architectures, but that does not adhere to a definition of strong incrementality. More recently, Kitaev et al. (2022) introduced a span-based model that incrementally encodes input sentences into discrete elements using vectors from GPT-2 mapped into a codebook. Despite this, it relied on bidirectional Transformers and a CYK architecture (Kitaev and Klein, 2018) for decoding these vectors into trees. Complementarily, Yang and Deng (2020) proposed an incremental decoder based on graph neural networks. Although they referred to their parser as strongly incremental, sentences were encoded with bidirectional architectures like BERT or XLNET (Devlin et al., 2019; Yang et al., 2019).

Incrementality has been also explored for other parsing formalisms. Stanojević and Steedman (2019) developed an almost fully incremental parser for combinatory categorical grammars (CCG), relying on ELMo embeddings (Peters et al., 2018) and a bidirectional LSTM for these predictions. Later, a genuinely fully incremental CCG parser was introduced (Stanojević and Steedman, 2020), using only ELMo's forward pass and a left-to-right LSTM, addressing biases in incremental CCG parsing. In the field of dependency parsing, incrementality has been a focus since the pre-neural era (Beuck and Menzel, 2013; Köhn and Menzel, 2014; Köhn and Baumann, 2016), with some models rivaling non-incremental ones. Recently, Ezquerro et al. (2023) found that with current neural architectures, incremental models for dependency parsing are less effective than bidirectional approaches. However, incorporating human-like reading strategies, such as brief delays, can significantly enhance performance, particularly in languages with leftward dependencies.

Contribution We study the viability and challenges of *fully incremental* constituent parsing with encoder-decoder architectures. All components strictly process the sentence from left to right, adding each read word to the partial tree based on the input prefix. For the encoder, we leverage generative LLMs. For the decoder, we reassess two options that generate partial trees based solely on current inputs: (i) an incremental parsing-as-tagging model (Gómez-Rodríguez and Vilares, 2018), and (ii) a transition-based decoder that uses graph-neural-network representations (Yang and Deng, 2020). The code is available at `https://github.com/anaezquerro/incpar`.

2 Incremental Constituent Parsing

Let $w = (w_1, ..., w_n)$ be a sequence of tokens such that $w_i \in \mathcal{V}$ for some vocabulary of tokens $\mathcal{V}$, we are interested in learning a function that can map w into a constituent tree T. Different from previous work, we are interested in modeling this function as an strictly incremental model. Under this setup, the decision at time step i is based only on the *prefix* $w_1...w_{i+k}$. It creates a partial tree, T_i, where each word w_i is added at its time step i, in a monotonic way. The delay parameter, k, mimics human reading processes, allowing for a slight look ahead to the upcoming words. Human parsing is believed to be very swift, with latencies as short as 250 milliseconds (Pulvermüller et al., 2009; Bemis and Pylkkänen, 2011). In this work, we will study both zero and small positive delays, i.e., $k \in [0, 2]$. Next, we review our encoders (§2.1) and decoders (§2.2).

2.1 Incremental encoders

The incremental encoder is a parameterized function Ψ_θ that produces a hidden representation vector $\mathbf{h}_i \in \mathbb{R}^h$ for each input token w_i based on its own prefix, thus $\mathbf{h}_i = \Psi_\theta(w_1...w_i)$. As for specific architectures, will rely on encoders both without and with pre-training. The former is a lower-bound baseline made of 4 stacked left-to-right LSTMs (Hochreiter and Schmidhuber, 1997). For the latter, we use multilingual GPT (mGPT; Shliazhko et al., 2022) and BLOOM-560M (Scao et al., 2022). mGPT has pre-training data for all languages studied, while BLOOM does not. This lets us measure the impact of: (i) no pre-training data, (ii) pre-training data for all languages, and (iii) missing pre-training data for some languages (see also §3).

2.2 Incremental decoders

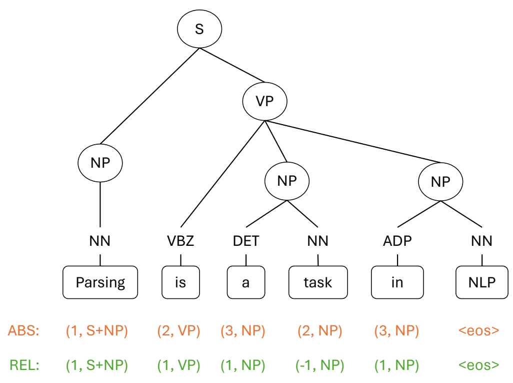
We propose two different architectures to implement our incremental decoders. In both cases, an intermediate module was added between the encoder and decoder to add prefix information up to word w_{i+k}. At each timestep i, this module accepts the encoder representations $\mathbf{h}_i...\mathbf{h}_{i+k}$ and generates a new delayed contextualization $\overline{\mathbf{h}}_i$ using a feed-forward network ($\overline{\mathbf{h}}_i = \mathbf{FFN}(\mathbf{h}_i...\mathbf{h}_{i+k})$). The delayed sequence $\overline{\mathbf{H}} = (\overline{\mathbf{h}}_1..., \overline{\mathbf{h}}_n)$ is directly passed as input to the decoder. Thus, these decoders produce an extra piece of the output tree based strictly on the prefix $w_1...w_{i+k}$.

On the one hand, we use decoders rooted in sequence labeling parsing (Gómez-Rodríguez and Vilares, 2018). Here, at each time step, each representation is mapped to a partial label that encodes a segment of the constituent tree primarily based on the preceding prefix. On the other hand, we choose the incremental decoder by Yang and Deng (2020). They use a graph neural network to contextualize the partial tree and make a decision (transition) at each time step based on the read token.

2.2.1 Incremental decoding as tagging

Given a sequence of delayed word contextualizations $\overline{\mathbf{H}} = (\overline{\mathbf{h}}_1...\overline{\mathbf{h}}_n)$, a tagging-based decoder maps each contextualization $\overline{\mathbf{h}}_i$ to a label $\ell_i \in \mathcal{L}$ and defines an injective and complete function to delinearize the sequence of labels into a valid constituent tree. Following Gómez-Rodríguez and Vilares (2018), each label is a tuple of the form ℓ_i=$(d_i, c_i) \in \mathcal{L}$, where d_i encodes a number l_i, the total number of levels in common between w_i and w_{i+1}, and c_i encodes the lowest non-terminal in common between those two words.[1] l_i can be encoded in d_i either directly ($d_i = l_i$, absolute encoding) or as a difference from the previous value

[1]The encoding is injective and complete for constituent trees without unary chains. The specifics can be found in the reference paper. Here, unary chains were collapsed in a single artificial constituent and recovered in the decoding step.



S
VP
NP
NP
NP
NN
VBZ
DET
NN
ADP
NN
Parsing
is
a
task
in
NLP
ABS: (1, S+NP) (2, VP) (3, NP) (2, NP) (3, NP) <eos>
REL: (1, S+NP) (1, VP) (1, NP) (-1, NP) (1, NP) <eos>

Figure 1: Absolute (orange) and relative (green) indexing from Gómez-Rodríguez and Vilares (2018). Note that unary chains are collapsed in an artificial constituent (first label). The final label indicates the end of sentence.

($d_1 = l_1$ and $d_i = l_i - l_{i-1}$ for $i \geq 2$, relative encoding). See Figure 1 for an example.

We chose this encoder over newer sequence labeling linearizations that have been recently published, such as tetra-tagging (Kitaev and Klein, 2020) and shift-reduce parsing through pre-order, post-order, and in-order linearizations (Amini and Cotterell, 2022). We did so due to a few practical reasons: (i) it is more user-friendly with existing libraries for transforming constituent trees into label sequences; and (ii) it accommodates non-binary trees, like the juxtapose model (binarizing and unbinarizing is trivial, yet necessary for these mentioned alternatives).[2]

That said, our decoder is straightforward. Given an incrementally delayed contextualized input $\overline{\mathbf{H}} = (\overline{\mathbf{h}}_1, ..., \overline{\mathbf{h}}_n)$, each label ℓ_i is computed as $\mathbf{FFN}_\ell(\overline{\mathbf{h}}_i)$, where $\mathbf{FFN}_\ell$ is a 1-layered feed-forward network with a softmax activation.

2.2.2 Incremental decoding as transition-based parsing

Similar to the tag-based decoders, transition-based systems incrementally process each word contextualization to generate a sequence of actions of variable length m. Each action updates the system's inner representation of the partial tree until the sequence is fully processed and the final state retrieves the complete predicted tree. As a transition-based decoder, we use the strong incremental decoder by Yang and Deng (2020). It generates a sequence of n transitions, adding exactly one token to the partial tree at each time step. Namely, each time step is represented by a partial tree T_{i-1}, which it is updated based on the subsequent w_i and the rightmost chain of T_{i-1} (denoted as $\mathcal{R}(T_{i-1})$)[3] by performing one of these actions:

- *attach*$(\varphi^{\text{tgt}}, \varphi^{\text{prt}})$: Attaches a new subtree to $\mathcal{R}(T_{i-1})$. It creates a non-terminal parent node φ^{prt} and puts the w_i as its terminal node. φ^{prt} also becomes the rightmost child of an existing non-terminal node $\varphi^{\text{tgt}} \in \mathcal{R}(T_{i-1})$.
- *juxtapose*$(\varphi^{\text{tgt}}, \varphi^{\text{prt}}, \varphi^{\text{new}})$: Replaces the non-terminal node $\varphi^{\text{tgt}} \in \mathcal{R}(T_{i-1})$ with the node φ^{new}. φ^{tgt} takes the role of left child of φ^{new} (keeping its descendants). The right child of φ^{new} is a fresh subtree rooted at φ^{prt} with the new read word w_i as only child.

Given a partial tree T_{i-1}, each span extended from fencepost $l-1$ to r is represented according to Equation 1 as a concatenation of (i) an embedding of the non-terminal symbol of the span ($\mathbf{c}_{l,r}$), and (ii) an embedding corresponding to the difference of the positions $\mathbf{p}_l$ and $\mathbf{p}_r$. All the spans of the partial tree T_{i-1} are stacked together in a matrix $\mathbf{X}_i = [\mathbf{C}_i, \mathbf{P}_i]$ and then passed through a graph convolutional network (GCN) to obtain a new contextualized matrix $\tilde{\mathbf{X}}_i = [\tilde{\mathbf{C}}_i, \tilde{\mathbf{P}}_i]$, where each row vector $\tilde{\mathbf{x}}_{l,r}$ is split as $\tilde{\mathbf{x}}_{l,r} = [\tilde{\mathbf{c}}_{l,r}, \tilde{\mathbf{p}}_{l,r}]$ using the same input dimensions (see Equation 1) to separate positional from constituent information. Given the contextualization of a new input word $\overline{\mathbf{h}}_i$ with its positional embedding $\mathbf{p}_i$, the scores to select the target node $\mathbf{s}_i^{\text{tgt}}$ are computed by two FFNs which operate with those word and span representations in the rightmost chain, denoted as $\tilde{\mathbf{X}}_i^{\mathcal{R}} = [\tilde{\mathbf{C}}_i^{\mathcal{R}}, \tilde{\mathbf{P}}_i^{\mathcal{R}}]$ (Equation 2). Finally, the scores for the parent and new nodes ($\mathbf{s}_i^{\text{prt}}$ and $\mathbf{s}_i^{\text{new}}$) are generated from $\overline{\mathbf{h}}_i$ and $\mathbf{p}_i$ vectors with the weighted representation of the rightmost chain (Equation 3).

$$\mathbf{x}_{l,r} = [\mathbf{c}_{l,r}, (\mathbf{p}_r - \mathbf{p}_l)/2] \tag{1}$$

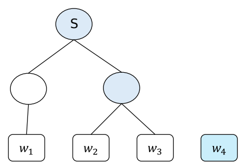
$$\mathbf{s}_i^{\text{tgt}} = \text{FFN}_c([\tilde{\mathbf{C}}_i^{\mathcal{R}}, \overline{\mathbf{h}}_i]) + \text{FFN}_p([\tilde{\mathbf{P}}_i^{\mathcal{R}}, \mathbf{p}_i]) \tag{2}$$

$$\mathbf{s}_i^{\text{prt}}, \mathbf{s}_i^{\text{new}} = \text{FFN}\Big([\overline{\mathbf{h}}_i, \mathbf{p}_i, (\mathbf{s}_i^{\text{tgt}} \tilde{\mathbf{X}}_i^{\mathcal{R}})]\Big) \tag{3}$$

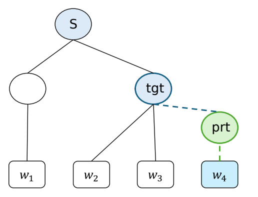
[2]Also, even if Kitaev and Klein and Amini and Cotterell report better results, it is worth noting that the original papers cannot be directly compared in terms of results due to different implementations. For instance, Gómez-Rodríguez and Vilares relied on LSTMs and a simple decoder based on feed-forward networks, while the tetra-tagging paper used BERT and did not employ a sequence labeling decoder, but rather an efficient and simple dynamic programming approach.

[3]Formally, the rightmost chain of a tree T_{i-1} is defined by the set of non-terminal nodes whose rightmost fencepost coincides with the last word of the sentence (see Figure 2).

(a) Partial tree T_3 built upon word w_3.



(b) *attach*(tgt, prt).



(c) *juxtapose*(tgt, prt, new).

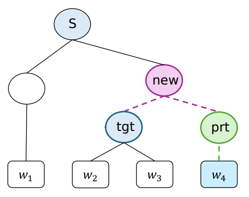


Figure 2: Transitions defined by Yang and Deng (2020) for a partial tree T_3 when a new word w_4 is added. Nodes in $\mathcal{R}(T_3)$ are marked in blue color.

Figure 2 shows the update of a partial tree T_3 (Figure 2a) when applying the *attach* (Figure 2b) or *juxtapose* (Figure 2c) actions. Note that the target node always belongs to the rightmost chain and at least one non-terminal node is added at each time step, producing always a valid partial tree T_i.[4]

3 Experiments

Setup To create our models, we used the `supar`[5] library as our starting point. It implements non-incremental parsers for the main parsing formalisms, including constituent parsing, and allows for plug-and-play integration of most large language models, including generative ones. For additional information, see Appendix A.2.

Data We use both the English Penn Treebank (Marcus et al., 1993) and the set of multilingual treebanks released as a part of the SPMRL shared task (Seddah et al., 2013).[6]

Metrics We use labeled bracketing F1-score, with the `COLLINS.prm` (for PTB) and `evalb_spmrl.prm` (for SPMRL) files.[7]

Upper-bound baselines We compare our models against counterparts that are not fully incremental. On the one hand, we consider Kitaev and Klein (2018)'s approach as an upper-bound baseline, as it uses Transformers and a powerful CYK neural decoding method. On the other hand, we explore partially incremental versions of our strong incremental models as control baselines, where the encoder is replaced with a bidirectional encoder, specifically XLM-RoBERTa (Conneau et al., 2020).

3.1 Results

Table 1 presents the outcomes for the strict incremental models with $k = 0$, compared to the upper-bound and control parsers. The results suggest that the main challenges in competing with bidirectional systems are primarily associated with the encoder side. This finding is similar to observations made by other authors for different paradigms, such as dependency parsing, as noted by Ezquerro et al. (2023). Particularly, we observe in Table 1 that models equipped with an incremental decoder and a non-incremental encoder (the control columns) achieve near state-of-the-art results. However, the F1-score substantially diminishes when switching to an incremental encoder. Across encoders, mGPT performs best overall. For languages not included in its pre-training data, BLOOM's performance is closer (yet usually higher) to that of the LSTM encoders, but it always performs worse than mGPT. We also observe clear differences across decoders. The transition-based decoders, while performing on average 10 points below the upper bound model, yield reasonable representations. On the other hand, the incremental sequence-labeling decoders achieve a subpar F1 score, on average 27 points below state-of-the-art parsers and 17 points below the transition-based decoder.

Table 2 compares our incremental models with zero delay to counterpart versions with delays of 1 and 2. The improvements are noticeable in both decoders, especially from delay zero to one. On average, for the sequence labeling decoder, moving from delay zero to one improves performance by 13.7 and 15.6 percentage points for the LSTM and mGPT encoders, respectively. Meanwhile, the improvements from delay 1 to delay 2 show clear diminishing returns, with only a 0.8 and 2.3 point improvement. The trend is similar for the transition-based decoder. When setting k=1, it shows average improvements of 8.2 points (using the vanilla LSTM encoder) and 4.5 points (mGPT) compared to the strict incremental version. However, there is

[4] Yang and Deng (2020) proved that the attach-juxtapose is injective for constituent trees without unary chains.

[5] `https://parser.yzhang.site/`

[6] We do not report results for the Arabic treebank since it requires a paid license to be used.

[7] BLOOM lacks pre-training data for German, Hungarian, Hebrew, Swedish, Polish, and Korean. As mentioned earlier, this is still useful to gather additional comprehension on how an incremental parser with a generative LLM performs on languages it was not specifically pre-trained for.

	Incremental						Control		KK
	SL			TB			SL	TB	
●●● en	$40.4^{\rightarrow}_{\blacktriangle a}$	$54.4^{\circledast}_{\blacktriangle r}$	$57.4^{\diamondsuit}_{\blacktriangle r}$	$77.2^{\rightarrow}_{\cdots}$	$83.5^{\circledast}_{\cdots}$	$85.7^{\diamondsuit}_{\cdots}$	$93.1^{\infty}_{\blacktriangle r}$	$94.5^{\infty}_{\cdots}$	95.5
●●● eu	$59.0^{\rightarrow}_{\blacktriangle r}$	$60.1^{\circledast}_{\blacktriangle r}$	$64.1^{\diamondsuit}_{\blacktriangle r}$	$71.4^{\rightarrow}_{\cdots}$	$76.5^{\circledast}_{\cdots}$	$81.8^{\diamondsuit}_{\cdots}$	$91.1^{\infty}_{\blacktriangle a}$	$92.8^{\infty}_{\cdots}$	93.6
●● de	$34.6^{\rightarrow}_{\blacktriangle r}$	$46.3^{\circledast}_{\blacktriangle a}$	$52.5^{\diamondsuit}_{\blacktriangle a}$	$51.9^{\rightarrow}_{\cdots}$	$67.4^{\circledast}_{\cdots}$	$72.9^{\diamondsuit}_{\cdots}$	$90.7^{\infty}_{\blacktriangle a}$	$91.7^{\infty}_{\cdots}$	88.9
●●● fr	$39.7^{\rightarrow}_{\blacktriangle a}$	$50.2^{\circledast}_{\blacktriangle r}$	$53.8^{\diamondsuit}_{\blacktriangle r}$	$64.9^{\rightarrow}_{\cdots}$	$71.7^{\circledast}_{\cdots}$	$74.5^{\diamondsuit}_{\cdots}$	$86.0^{\infty}_{\blacktriangle a}$	$86.6^{\infty}_{\cdots}$	91.5
●● he	$66.2^{\rightarrow}_{\blacktriangle a}$	$66.4^{\circledast}_{\blacktriangle r}$	$76.1^{\diamondsuit}_{\blacktriangle r}$	$65.4^{\rightarrow}_{\cdots}$	$74.3^{\circledast}_{\cdots}$	$84.4^{\diamondsuit}_{\cdots}$	$91.8^{\infty}_{\blacktriangle a}$	$93.8^{\infty}_{\cdots}$	92.8
●● hu	$72.0^{\rightarrow}_{\blacktriangle r}$	$69.3^{\circledast}_{\blacktriangle r}$	$76.6^{\diamondsuit}_{\blacktriangle r}$	$69.8^{\rightarrow}_{\cdots}$	$82.2^{\circledast}_{\cdots}$	$89.1^{\diamondsuit}_{\cdots}$	$94.5^{\infty}_{\blacktriangle a}$	$95.3^{\infty}_{\cdots}$	96.3
●● ko	$63.8^{\rightarrow}_{\blacktriangle r}$	$63.7^{\circledast}_{\blacktriangle r}$	$70.4^{\diamondsuit}_{\blacktriangle r}$	$75.7^{\rightarrow}_{\cdots}$	$77.7^{\circledast}_{\cdots}$	$81.9^{\diamondsuit}_{\cdots}$	$89.0^{\infty}_{\blacktriangle r}$	$89.8^{\infty}_{\cdots}$	91.9
●● pl	$71.6^{\rightarrow}_{\blacktriangle a}$	$71.8^{\circledast}_{\blacktriangle r}$	$79.7^{\diamondsuit}_{\blacktriangle a}$	$77.6^{\rightarrow}_{\cdots}$	$84.7^{\circledast}_{\cdots}$	$91.4^{\diamondsuit}_{\cdots}$	$96.2^{\infty}_{\blacktriangle a}$	$96.8^{\infty}_{\cdots}$	97.1
● sv	$47.6^{\rightarrow}_{\blacktriangle r}$	$47.3^{\circledast}_{\blacktriangle r}$	$60.3^{\diamondsuit}_{\blacktriangle r}$	$60.4^{\rightarrow}_{\cdots}$	$64.1^{\circledast}_{\cdots}$	$78.2^{\diamondsuit}_{\cdots}$	$87.6^{\infty}_{\blacktriangle a}$	$90.2^{\infty}_{\cdots}$	92.0
μ	55.0	58.8	65.7	68.3	75.8	82.2	91.1	92.4	93.3

Table 1: Labeled F-score paired with best sequence labeling (SL) and transition-based (TB) decoder. μ represents macro average results. Superscripts denote the encoder choice: LSTM ($\rightarrow$), BLOOM-560M ($\circledast$), mGPT ($\diamondsuit$), XLM-RoBERTa (∞). Subscripts denote the decoder configuration: absolute (a), relative (r), GCN ($\cdots$) and FFN ($\blacktriangle$). The upper bound baseline performance (**KK**, (Kitaev and Klein, 2018)) is also included. Language codes come from ISO 639-1 and left colored dots indicate the pretraining availability in LMs.

	SL				TB			
	LSTM ($\rightarrow$)		MGPT ($\diamondsuit$)		LSTM ($\rightarrow$)		MGPT ($\diamondsuit$)	
●●● en	$68.3_{27.9}$	$72.4_{32.0}$	$82.3_{24.9}$	$86.1_{28.7}$	$83.4_{6.2}$	$84.2_{7.0}$	$90.9_{5.2}$	$91.6_{5.9}$
●●● eu	$78.0_{19.0}$	$77.2_{18.2}$	$84.8_{20.7}$	$86.8_{22.7}$	$81.7_{10.3}$	$81.2_{9.8}$	$87.1_{5.3}$	$88.0_{6.2}$
●● de	$57.5_{22.9}$	$59.0_{24.4}$	$72.8_{20.3}$	$76.4_{23.9}$	$64.6_{12.7}$	$64.5_{12.6}$	$81.3_{8.4}$	$83.3_{10.4}$
●●● fr	$59.0_{19.3}$	$60.4_{20.7}$	$75.2_{21.4}$	$78.5_{24.7}$	$73.5_{8.6}$	$76.4_{11.5}$	$81.0_{6.5}$	$83.0_{8.5}$
●● he	$75.7_{9.5}$	$76.3_{10.1}$	$84.7_{8.6}$	$85.5_{9.4}$	$77.7_{12.3}$	$79.7_{14.3}$	$87.5_{3.1}$	$88.0_{3.6}$
●● hu	$76.4_{4.4}$	$79.6_{7.6}$	$84.8_{8.2}$	$87.5_{10.9}$	$82.1_{12.3}$	$85.2_{15.4}$	$92.0_{2.9}$	$92.1_{3.0}$
●● ko	$70.0_{6.2}$	$70.0_{6.2}$	$78.0_{7.6}$	$80.0_{9.6}$	$77.1_{1.4}$	$77.8_{2.1}$	$83.9_{2.0}$	$84.6_{2.7}$
●● pl	$83.1_{11.5}$	$82.0_{10.4}$	$91.4_{11.7}$	$92.4_{12.7}$	$86.2_{8.6}$	$87.8_{10.2}$	$93.6_{2.2}$	$94.2_{2.8}$
● sv	$64.3_{16.7}$	$62.9_{15.3}$	$77.3_{17.0}$	$79.1_{18.8}$	$70.1_{9.7}$	$66.9_{6.5}$	$82.8_{4.6}$	$83.8_{5.6}$
μ	$63.2_{13.7}$	$64.0_{14.5}$	$81.3_{15.6}$	$83.6_{17.9}$	$69.6_{8.2}$	$70.4_{9.0}$	$86.7_{4.5}$	$87.6_{5.4}$

Table 2: LF scores with delay 1 and 2 (first and second subcolumn) Notation as in Table 1. Subscripts denote performance boost over zero-delay fully incremental results from Table 1.

only a 0.8 and a 0.9 point improvement compared to the models with delay one. These diminishing returns indicate that small delays are not enough to close the gap, and strategies to improve incremental encoders such as prophecy tokens that can simulate larger delays (Madureira and Schlangen, 2020) may be needed - although tailored for parsing and contemporary language models.

Finally, some phrases may be more ambiguous than others in an incremental setting due to factors such as sentence structure, word order, or semantics. Figure 3 shows F1-scores for the most common non-terminals in diverse languages: English, Hebrew, Basque, and Korean, for models with $k \in [0, 1, 2]$. For space reasons, we include

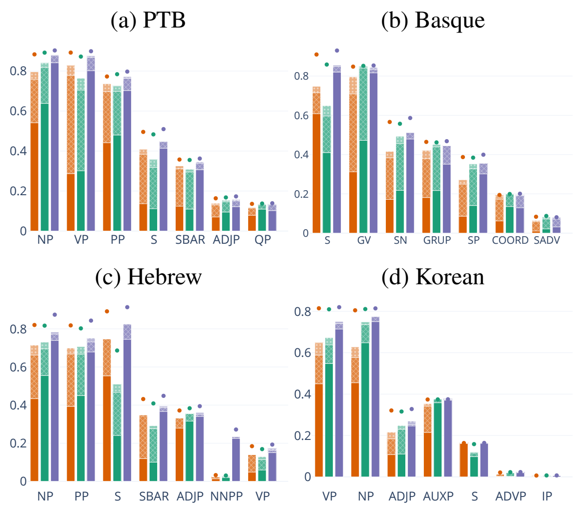


Figure 3: F-Score of absolute (orange), relative (green) and transition-based (purple) decoders with mGPT (bars) and XLM-RoBERTa (dots) encoders per constituent. Different textures are used for delay 0 (solid), 1 (dotted) and 2 (gridded).

only the most coarse-grained non-terminals. Unary chains are excluded. Across the board, positive delays, especially $k = 1$, have a much greater impact on sequence-labeling decoders, particularly benefiting longer span types like noun, verb, and prepositional phrases (span lengths are in Appendix Table 3). Also, behaviors across non-terminals and languages can vary greatly with incrementality, e.g., while delay is crucial for phrases such as VB for PTB or SBAR for Hebrew, its need is negligible for others such as Hebrew ADJP.

4 Conclusion

This paper introduced a set of strictly incremental encoder-decoder constituent parsers, using generative language models and two types of decoders: one based on parsing as tagging, and the other on transition-based parsing with partial graph neural network representations. We tested the models in a diverse multilingual setting and also simulated human reading processes with positive delays of a few upcoming words. The results suggest that a significant portion of future challenges may be centered on the encoding side, and in how different phrases might be affected by the absence of bidirectionality. In this context, exploring research lines to inform the decoder, such as speculative real-time generation of next tokens in real time, could be a valuable step to explore parsing methods closer to human reading processes.

Acknowledgments

We thank the anonymous reviewers for their very complete and useful suggestions. This work has received funding by the European Research Council (ERC), under the Horizon Europe research and innovation programme (SALSA, grant agreement No 101100615), ERDF/MICINN-AEI (SCANNER-UDC, PID2020-113230RB-C21), Xunta de Galicia (ED431C 2020/11), Grant GAP (PID2022-139308OA-I00) funded by MCIN/AEI/10.13039/501100011033/ and by ERDF "A way of making Europe", Cátedra CICAS (Sngular, University of A Coruña), and Centro de Investigación de Galicia "CITIC", funded by the Xunta de Galicia through the collaboration agreement between the Consellería de Cultura, Educación, Formación Profesional e Universidades and the Galician universities for the reinforcement of the research centres of the Galician University System (CIGUS).

Limitations

Non-monotonicity Our definition of incrementality applies exclusively to monotonic parsers. In cases of non-monotonicity, a parser might abandon its existing partial output and revise it as new information comes in. This is carefully discussed in (Ezquerro et al., 2023) for dependency parsing. Similarly, we chose to focus solely on monotonic constituent parsers. First, our goal is to maintain a straightforward implementation that is on par with others, avoiding the added complexity that repair strategies entail. Second, dealing with non-monotonicity requires thinking a thorough evaluation framework. In this respect, comparing against (partial) incremental parsers is challenging, as metrics must account for partial analysis, which is not accommodated by the standard bracketing F1-measure. In turn, such metrics on partial analyses are meaningless for non-incremental parsers, which often do not even produce any partial outputs, precluding direct comparison against them.

Discontinuous constituent parsing We restricted our analysis to continuous constituent parsing and observed that modern incremental parsers still exhibit shortcomings in this area. Studying the impact of incrementality on discontinuities, i.e., discontinuous spans within a sentence that form specific constituents, presents a more challenging aspect of constituent parsing. This phenomenon is particularly observed in languages with free word order. In this regard, there are several avenues to explore. For example, we could draw inspiration from the incremental transition-based algorithm described by Coavoux and Crabbé (2017), or the sequence labeling approach suggested by Vilares and Gómez-Rodríguez (2020), which shows potential to be adapted to an incremental setup.

Experiments on lower-resourced languages Unlike in other paradigms like dependency parsing, the availability of a diverse range of treebanks spanning various typologies is more limited for constituent parsing. We used the treebanks presently at our disposal, which include the English Penn Treebank and the SPMRL treebanks. However, it is worth noting that we were unable to access the Arabic dataset due to its paid license. Yet, our experiments consider: English (Indo-European, Germanic), Basque, German (Indo-European, Germanic), French (Indo-European, Romance), Hebrew (Afro-Asiatic, Semitic), Hungarian (Uralic, Ugric), Korean, Polish (Indo-European, Slavic) and Swedish (Indo-European, Germanic).

Availability of multilingual large language models Research on generative LLMs is extensive, and many models are being released contemporaneously with this paper. However, highly multilingual versions are more rare. Two main available resources are BLOOM and mGPT, both of which we evaluated. This double evaluation allowed us to establish differences in performance, particularly in terms of incrementality, depending on whether the models contained pre-training data for a given target language or not.

Computational capabilities We lacked access to extensive computing infrastructure or a budget for cloud-based scaling that would allow us to fine-tune more powerful multilingual language models such as the LLaMa family. We had access to NVIDIA GeForce RTX 3090 servers (each with 24GB) and one NVIDIA A100 GPU (with 80GB). We managed to fine-tune the smallest BLOOM language model (560M parameters) within reasonable time frames. Although we could technically fit the 3B version on the A100 GPU with a minimal batch size, the impractical training duration made it infeasible for a comprehensive multilingual study like the one proposed in this work.

References

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.

Douglas K. Bemis and Liina Pylkkänen. 2011. Simple composition: A magnetoencephalography investigation into the comprehension of minimal linguistic phrases. *Journal of Neuroscience*, 31(8):2801–2814.

Niels Beuck and Wolfgang Menzel. 2013. Structural prediction in incremental dependency parsing. In *Computational Linguistics and Intelligent Text Processing: 14th International Conference, CICLing 2013, Samos, Greece, March 24-30, 2013, Proceedings, Part I 14*, pages 245–257. Springer.

Maximin Coavoux and Benoît Crabbé. 2017. Incremental discontinuous phrase structure parsing with the GAP transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale.

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing.

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*, pages 52–66, Nusa Dua, Bali. Association for Computational Linguistics.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR.

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.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

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

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.

Nikita Kitaev, Thomas Lu, and Dan Klein. 2022. Learned incremental representations for parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3086–3095, Dublin, Ireland. Association for Computational Linguistics.

Arne Köhn. 2018. Incremental natural language processing: Challenges, strategies, and evaluation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2990–3003, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Arne Köhn and Timo Baumann. 2016. Predictive incremental parsing helps language modeling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 268–277, Osaka, Japan. The COLING 2016 Organizing Committee.

Arne Köhn and Wolfgang Menzel. 2014. Incremental predictive parsing with TurboParser. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 803–808, Baltimore, Maryland. Association for Computational Linguistics.

Ioannis Konstas, Frank Keller, Vera Demberg, and Mirella Lapata. 2014. Incremental semantic role labeling with Tree Adjoining Grammar. In *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP), pages 301–312, Doha, Qatar. Association for Computational Linguistics.

Brielen Madureira and David Schlangen. 2020. Incremental processing in the age of non-incremental encoders: An empirical assessment of bidirectional models for incremental NLU. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 357–374, Online. 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.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Friedemann Pulvermüller, Yury Shtyrov, and Olaf Hauk. 2009. Understanding in an instant: Neurophysiological evidence for mechanistic language circuits in the brain. *Brain and Language*, 110(2):81–94.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

Yikang Shen, Shawn Tan, Alessandro Sordoni, Siva Reddy, and Aaron Courville. 2021. Explicitly modeling syntax in language models with incremental parsing and a dynamic oracle. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1660–1672, Online. Association for Computational Linguistics.

Oleh Shliazhko, Alena Fenogenova, Maria Tikhonova, Vladislav Mikhailov, Anastasia Kozlova, and Tatiana Shavrina. 2022. mgpt: Few-shot learners go multilingual.

Miloš Stanojević and Mark Steedman. 2019. CCG parsing algorithm with incremental tree rotation. 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 228–239, Minneapolis, Minnesota. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 2020. Max-margin incremental CCG parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4111–4122, Online. Association for Computational Linguistics.

Ivan Titov and James Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks. *Advances in Neural Information Processing Systems*, 33:21687–21698.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

A Appendix

A.1 Treebank statistics

Table 3 shows the frequency and average length (defined as the difference between initial and final fencepost) of the constituents displayed in Figure 3.

A.2 Hyperparameters configuration

Tables 4 and 5 show the configuration of the models and the training hyperparameters for each encoder type. In pretrained models, each sentence $(w_1, ..., w_n)$ was passed through all encoder layers to compute the last hidden state $(\mathbf{e}_1, ..., \mathbf{e}_n)$ and

en	NP	VP	PP	S	SBAR	ADJP	QP
λ	4.96	11.2	5.96	12.71	12.2	5.02	2.97
%	43.13	24.56	16.95	6.29	3.6	1.79	1.7
eu	S	GV	SN	GRUP	SP	COORD	SADV
λ	8.86	2.31	3.66	2.26	3.38	2.04	2.61
%	34.07	18.47	15.46	12.38	11.2	2.8	2.08
de	NP	PP	VN	SENT	COORD	VPINF	NC
λ	6.37	6.5	2.52	30.44	9.4	12.35	2.59
%	32.79	26.25	7.04	6.04	4.53	3.2	2.92
fr	NP	PP	S	VP	AP	PN	CNP
λ	3.42	3.69	8.5	5.81	3.09	2.16	4.72
%	28.66	25.55	21.22	9.16	4.37	2.98	2.75
he	NP	PP	S	SBAR	ADJP	NNPP	VP
λ	5.14	5.93	17.21	13.15	2.69	2.65	10.67
%	44.19	21.68	15.21	5.4	4.44	3.2	2.56
hu	NP	CP	ADJP	PP	XP	ADVP	V
λ	3.58	14.97	3.79	4.09	7.36	2.88	2.0
%	57.11	29.3	7.54	3.97	1.26	0.66	0.16
ko	VP	NP	ADJP	AUXP	S	ADVP	IP
λ	6.76	3.79	6.29	2.14	13.45	3.03	2.06
%	49.75	36.0	8.04	4.1	1.53	0.39	0.16
pl	FNO	ZDANIE	FPM	FWE	FZD	FPT	FORMACZAS
λ	3.61	8.09	3.3	3.57	8.4	3.9	2.04
%	34.18	28.15	19.79	5.39	3.56	3.34	2.21
sv	NP	S	PP	VP	XP	AP	AVP
λ	4.61	12.14	4.61	7.64	4.24	2.65	3.51
%	31.1	27.35	20.69	10.35	6.46	2.48	0.88

Table 3: Frequency (%) and average length (λ) of most frequent constituents of each treebank. Root and unary spans were removed.

then projected to a new reduced space of dimension h with a feed-forward network. The final reduced sequence $\mathbf{H} = (\mathbf{h}_1, ..., \mathbf{h}_n)$ is the one passed to the delay module. In the case of non-pretrained encoders, each word w_i was represented as a concatenation of (i) a word embedding of dimension h_w, (ii) the PoS tag embedding of dimension h_p and (iii) the last hidden state of a Character-LSTM (Dozat and Manning, 2017) of dimension h_c, resulting into a final input embedding $\mathbf{w}_i \in \mathbb{R}^{h_w+h_p+h_c}$. The input matrix $\mathbf{W} = (\mathbf{w}_1, .., \mathbf{w}_n)$ is introduced to the LSTM encoder (with randomly initialized weights) and its last hidden states $(\mathbf{h}_1, ..., \mathbf{h}_n)$ are passed through the delay module and the decoder. The decoder is a 3-layered Graph Convolutional Layer for the Attach-Juxtapose parser or a feed-forward network for the case of the sequence labeling decoder. The complete network was trained with the CrossEntropy loss function and AdamW as optimizer, adapting the batches to the model size. Dropout was set in both encoder and decoder and the best validation performance was finally retrieved.

Finally, Table 6 displays various estimates of inference speeds for different models.

Hyp.	**LLM**			**Non-pretrained**	
	XLM	BLOOM	mGPT	LSTM→	BiLSTM↔
word emb. (h_w)		-		300	
PoS emb. (h_p)		-		100	
char. emb.		-		50	
char. LSTM (h_c)		-		100	
# enc. layers		1		4	
enc. emb. (h)		100		400	
% enc. dropout		0.33		$0.33_{\text{sh.}}$	
# GCN layers		3		3	
# FFN layers		1		1	
% dec. dropout		0.33		$0.33_{\text{sh.}}$	

Table 4: Model configuration for pretrained and non-pretrained models. The number of encoder layers for LLMs refers to the number of last hidden states obtained for each word. LSTM-based encoders use the shared-dropout technique (Gal and Ghahramani, 2016) as described in Dozat and Manning (2017).

Hyp.	**LLM**			**Non-pretrained**	
	XLM	BLOOM	mGPT	LSTM→	BiLSTM↔
optimizer		AdamW		AdamW	
lr		5e-5		1e-3	
lr decay		linear (0.5)		exponential (0.1)	
epochs		30		200	
batch size	500	500	100	5000	

Table 5: Training hyperparameters for pretrained and non-pretrained models. AdamW is set as optimizer with $\beta_0 = 0.9$, $\beta_1 = 0.9$ and $\varepsilon = 10^{-12}$, and batch sampling is fixed to minimize sequence padding.

	SL (▲)				
	LSTM→	BiLSTM↔	BLOOM	mGPT	XLM
en	856.02	688.98	354.21	144.03	403.09
eu	2096.61	1265.82	424.77	168.95	523.03
fr	678.24	467.33	281.33	116.02	313.66
de	1348.32	969.85	306.75	155.42	417.9
he	497.8	673.3	192.07	122.14	323.24
hu	1258.99	809.41	233.1	124.26	390.06
ko	1838.16	1477.9	261.41	151.59	486.57
pl	2091.04	1725.57	439.35	242.28	631.53
sw	1465.52	1132.12	325.1	174.44	507.27
μ	1347.86	1023.36	313.12	155.46	444.04

	TB (•••)				
	LSTM→	BiLSTM↔	BLOOM	mGPT	XLM
en	191.21	188.75	146.69	90.83	152.34
eu	569.21	524.83	295.27	135.45	341.40
fr	109.87	106.04	90.20	62.09	94.29
de	249.92	241.20	160.34	102.48	190.38
he	164.55	155.95	94.11	72.38	130.94
hu	235.16	224.06	138.12	90.26	181.44
ko	570.62	517.57	200.15	128.72	306.19
pl	717.30	586.34	297.53	196.40	400.62
sw	331.67	286.10	196.45	116.33	229.96
μ	348.83	314.54	179.87	110.55	225.28

Table 6: Inference speed (in sentences per second) of the evaluated models across different languages. Symbols come from Table 1.