Sequence Labeling for Constituent Parsing: A Comparative Study and Encoding Innovations

Diego Roca, David Vilares, and Carlos Gómez-Rodríguez

Universidade da Coruña, CITIC, Department of Computer Science and Information Technologies
d.roca1@udc.es, david.vilares@udc.es, carlos.gomez@udc.es

Various encodings have been proposed to cast constituent parsing in terms of a sequence labeling task. However, unlike in the case of dependency parsing, existing comparisons have not been entirely homogeneous and, to the best of our knowledge, there is no systematic evaluation of these encodings under uniform configurations. A homogeneous evaluation needs to account for various aspects that could influence results, either by controlling for these aspects to ensure uniformity (e.g., network architecture, parameter settings, postprocessing of ill-formed output), or by systematically analyzing their impact (e.g., the impact of binary versus arbitrary structures). In this article, we: (1) compare different encodings comprehensively both theoretically and empirically, on a modern neural architecture and across nine languages, and (2) introduce new encodings and variants, including an encoding that our analysis finds particularly accurate and compact.

1. Introduction

Constituent parsing is the task of obtaining the syntax of a sentence in natural language represented as a phrase-structure tree. In this representation, the input words are the **terminal** (leaf) nodes. Their corresponding part-of-speech tags are often referred to as preterminal nodes (and often assumed to be precomputed). The intermediate nodes in the tree are referred to as **nonterminals**, and denote a given **constituent**. A constituent can be roughly defined as a span[1] of words that represent a valid phrase within the language. For instance, "*ate the apple*" is a verb phrase in English, "*the red dog*" a nominal phrase, and "*with great pain*" a prepositional phrase. There are two ways to structure constituent trees: either allowing each node to have an arbitrary branching factor (i.e., a parent constituent can have any number of children constituents) or restricting them to be k-ary trees (typically binary trees, i.e., $k = 2$). Figure 1 shows examples of these

[1] Usually a consecutive span of words, although in languages that exhibit free word order there can be discontinuous spans of words that form a valid constituent (Van Cranenburgh, Scha, and Bod 2016; Coavoux and Cohen 2019). In this work, we will focus strictly on continuous constituent parsing. To the best of our knowledge, only a single paper has addressed sequence labeling encodings for discontinuous structures (Vilares and Gómez-Rodríguez 2020), all based on variants of the same encoding, so there is no need for a comparison.

Action Editor: Giorgio Satta. Submission received: 30 May 2025; revised version received: 10 October 2025; accepted for publication: 11 December 2025.

https://doi.org/10.1162/COLI.a.603

Figure 1
Example of a constituent tree (a) and a valid right binarization of it (b).

two options. The distinction between these types typically depends on whether they have undergone preprocessing to limit the number of children a given nonterminal can have. It is important to note that one can convert from one type of representation to the other through the application of straightforward preprocessing and postprocessing heuristics.

Traditionally, constituent parsing relied on classical transition-based or chart-based algorithms. However, recent advances have shown that this problem can be effectively reinterpreted using more efficient paradigms, particularly by casting constituent parsing in terms of sequence labeling (Gómez-Rodríguez and Vilares 2018; Kitaev and Klein 2020; Amini and Cotterell 2022). This paradigm proposes linearizing a phrase-structure tree into a sequence of labels drawn from a predefined discrete set, assigning *exactly one* label to *each* word in the input sentence. A mapping that converts trees to sequences of labels in this way is called a sequence labeling **linearization** or **encoding**. This method is relevant both theoretically and practically, having garnered considerable interest in recent years, both for constituent parsing and other flavors of structured representations such as dependency grammar (Strzyz, Vilares, and Gómez-Rodríguez 2019b; Vacareanu et al. 2020; Amini, Liu, and Cotterell 2023).

Theoretically, while linear representations of constituent structures have a long history (e.g., the classic parenthesized representation of trees, or the sequence of transitions in a transition-based parser), recent work has proposed injective encodings that are specifically designed to integrate with sequence labeling frameworks, as they map trees to label sequences that have a strict one-to-one correspondence with the input words. This innovation addresses the theoretical limitations of previous parsing strategies, particularly those concerning high computational complexities and slow processing speeds, as it allows parsers to be built using highly optimized, off-the-shelf sequence labeling architectures that perform non-autoregressive inference in a single, highly parallelizable forward pass. This is in contrast to previous linearization approaches: While transition-based parsers (e.g., Liu and Zhang 2017) also define a linearization via sequences of transitions, and parenthesized linearizations have been used for sequence-to-sequence parsing (Vinyals et al. 2015), both of these require autoregressive decoding (each decision depends on the previous ones), which typically makes them slower and harder to parallelize on modern hardware.

Parsing as sequence labeling also offers several practical benefits. For instance, it removes the necessity for custom network architectures and complicated decoders that are dedicated solely to parsing tasks. Instead, it allows for the training of parsers using standard sequence labeling architectures, which are already extensively used for

various applications, such as named entity recognition, event extraction, or semantic role labeling (Yang and Zhang 2018). This adaptability substantially expands the range of tools available for generating structured outputs, enhancing the versatility and efficiency of parsing techniques, as well as lowering the barrier of entry for NLP practitioners that may not be familiar with ad hoc parsing algorithms, but are likely to know sequence labeling as a simple, widely-used method in NLP. In turn, the linearization of trees into sequences of labels that correspond in length to the input sentences makes syntactic information easier to use for downstream tasks. This transformation allows the linearized structures to be used as versatile embeddings, which find applications in various domains. Syntax injected as label embeddings was already shown useful to enhance performance in NLP tasks prior to the parsing as sequence labeling approach: For instance, in machine translation, Li et al. (2017) obtained good results using a non-injective (i.e., partial or lossy) encoding. Full-fledged sequence-labeling encodings can also be used in this way, with the additional advantage that they encode the whole tree, thereby enhancing performance in tasks where syntax is a key factor. This includes applications in semantic role labeling (Wang et al. 2019), enhancing low-resource learning (Rotman and Reichart 2019), named entity recognition (Alonso, Gómez-Rodríguez, and Vilares 2021), or solving math word problems (Wang et al. 2022). Furthermore, they are easily adaptable for use in multi-task learning setups to jointly learn multiple structured representations (Strzyz, Vilares, and Gómez-Rodríguez 2019a) without the need for ad hoc architectures.

Additionally, this approach has practical applications in the time of language models. For example, linearized parsing representations are commonly used to assess the probing frameworks of syntactic representations captured by pre-trained language models (Tenney et al. 2018; Lin, Tan, and Frank 2019; Vilares et al. 2020; Muñoz-Ortiz, Vilares, and Gómez-Rodríguez 2023; Kodner, Khalifa, and Payne 2023).

Finally, sequence-labeling parsing can be of interest for building psycholinguistically plausible parsing models: Not only does it lend itself to incremental parsing, but it is in line with recent psycholinguistic evidence pointing to syntactic processing being similarly distributed and deeply interconnected with lexico-semantic processing, rather than recruiting separate brain regions (Blank et al. 2016; Fedorenko et al. 2020), which would make models that solve parsing in the same way as tasks like tagging or entity recognition (and allowing for integration via multitask learning) promising with respect to ad-hoc, structurally separate parsing algorithms. In this respect, recent studies have demonstrated the instrumental role of the sequence labeling approach in the development of incremental natural language processing systems, as detailed by Ezquerro, Gómez-Rodríguez, and Vilares (2023, 2024). These systems aim to closely mimic human language processing, highlighting the extensive impact of this method in advancing our understanding of natural language.

In this scenario, a range of sequence labeling encodings have been proposed to convert constituent parsing into sequence labeling, with each encoding exploiting distinct characteristics of the tree structure. However, the published implementations of these encodings display variations in their decoding strategies. As a result, their performance as reported in the respective papers cannot, or arguably should not, be directly compared against one another. Factors contributing to this issue include non-standardized aspects like the prediction of binary versus arbitrary structures, as well as different neural architectures, language models, or parameters used in the taggers. Additionally, minor modifications in the encodings can have significant impacts, as can slight differences in the decoding algorithms. Challenges may also arise when integrating these encodings into existing off-the-shelf sequence labeling architectures.

These factors collectively underscore the complexity of directly comparing performance across different encoding strategies. That said, this article's contributions are four-fold. First, we unify existing encoding families under a standardized label format and conduct a systematic review of their features, strengths, and weaknesses from a theoretical perspective. Second, we introduce a new, accurate unbounded encoding that provides a compact label space and particularly good empirical results compared with other encodings. Third, we recast and implement the attach-juxtapose transition-based parser as a sequence-labeling parser for the first time. Fourth, we conduct a fair and homogeneous empirical evaluation of these encodings, using a uniform experimental setup, to analyze their performance on a range of treebanks for nine languages.

The source code for the implementation of all the encodings is available at: `https://github.com/Polifack/CoDeLin`.

2. Preliminaries

We next contextualize our work by: (i) reviewing the main off-the-shelf architectures used in NLP for sequence labeling tasks; (ii) contextualizing early approaches in shallow and partial parsing and connecting them to current methods where whole syntax trees are encoded as label sequences; and (iii) reviewing the principal work on constituent parsing as sequence labeling, as well as other syntactic formalisms of interest.

Brief Review of Off-the-Shelf Sequence Labeling Architectures for NLP Tasks. Sequence labeling is a structured prediction task where, given a source sequence, a model produces exactly one label for each input token. This approach is noteworthy because it restricts the number of inferences to a minimum, equal to the length of the input sequence. Note that this differs from sequence-to-sequence approaches, in which the mapping between input and output tokens does not maintain a one-to-one correspondence. Popular architectures for sequence labeling tasks include models such as Conditional Random Fields (Lafferty et al. 2001), which capture dependencies between output labels; Recurrent Neural Networks and Long Short-Term Memory networks (LSTMs; Hochreiter and Schmidhuber 1997), which process tokens sequentially and capture contextual information; and Transformers (Vaswani et al. 2017), which capture long-range dependencies and parallelize processing, improving efficiency and scalability for longer sequences. Particularly for Transformers, it is common to enhance performance by pre-training the architecture on language modeling tasks (Devlin et al. 2019), followed by fine-tuning for specific applications. In our work, we will focus specifically on these latter pre-trained transducers, as non-deep learning models are unable to learn the rich, contextualized vector representations that are key to capturing complex syntactic patterns, and neural models without pre-training perform well below state-of-the-art levels (Vilares et al. 2020).

Sequence Labeling for Shallow Parsing Tasks. Canonical work on sequence labeling for syntax-based representations dates back to the early 1990s, where NLP tasks such as chunking (Abney 1992; Ramshaw and Marcus 1995), also known as shallow phrase parsing, were widely popular. Chunking is about identifying subsequences of words that form non-overlapping linguistic groups in a flat and non-hierarchical manner, as in the sentence $[\textit{The quick brown fox}]_{NP}$ $[\textit{jumps}]_{VP}$ $[\textit{over}]_{PP}$ $[\textit{the lazy dog}]_{NP}$, where we could identify the noun phrase "The quick brown fox" or the verb phrase "jumps", among others. Mapping these chunks to sequence labeling representations was straightforward, and standard BIO schemes could be used for such purposes (Ramshaw and Marcus

1995). Many other tasks—e.g., part-of-speech (PoS) tagging (Brants 2000), semantic role labeling (Màrquez et al. 2005), or Arabic diacritization (Schlippe, Nguyen, and Vogel 2008)—have been traditionally cast in terms of sequence labeling for similar reasons: efficiency, simplicity, and the availability of a wide spectrum of off-the-shelf tools that can be used to train systems without being an expert in the field. However, chunking is considerably simpler than other parsing tasks, such as constituent parsing. Additionally, at that time, linearizations that would allow framing such tasks as sequence labeling were lacking.

Thus, for more complex syntax-based tasks, the use of sequence labeling approaches has been infrequent until recently. We attribute this issue to two main reasons: (i) the inability of models to accurately represent long contexts, and (ii) the (aforementioned) absence of linearizations that could be effectively learned by these models.

As a notable exception, sequence labeling has been used for syntactic processing in the context of supertagging (Joshi and Srinivas 1994; Clark 2002; Zhang, Matsuzaki, and Tsujii 2009; Vaswani et al. 2016). Supertagging involves a rich tag representation, where each category encodes highly specific syntactic information, such as the types of sentences or directionality of arguments. It is typically used with the goal of improving parsing speed under complex grammatical formalisms where efficiency is an issue, having originated with lexicalized TAG (Joshi and Srinivas 1994) and then extended to other computationally challenging formalisms like CCG (Clark and Curran 2004) or HPSG (Ninomiya et al. 2006); and still being an active area of research today (Margueritte, Bekki, and Mineshima 2023; Zamaraeva and Gómez-Rodríguez 2024). Still, supertagging is a partial form of parsing, as supertags carry syntactic information but they do not encode a full parse tree—a parsing algorithm is still needed to use this information to build a complete parse.

The Path Towards Sequence Labeling for Full Parsing Tasks. The first attempt to use sequence labeling for full syntactic parsing was the work by Spoustová and Spousta (2010), who experimented with a sequence labeling linearization for full dependency parsing. In this linearization, the position of each word's syntactic head was encoded as a PoS tag-based offset, counting the number of previous/upcoming words (depending on whether the dependency is rightward or leftward) between the dependent and the head with the same PoS tag as the head. However, their results were quite modest, likely due to the limitations of sequence-labeling architectures at the time (especially in terms of effectively modeling long contexts), and the comparison against contemporary approaches was also very limited.

Thus, most of the work on sequence labeling parsing for different formalisms has risen after the adoption of deep neural networks and, in particular, from 2018 onwards. In this respect, it is worth remarking that sequence labeling parsing is not equivalent to the related approach of sequence-to-sequence parsing. Sequence-to-sequence architectures appeared a few years earlier, stemming from the work by Sutskever, Vinyals, and Le (2014) using bidirectional LSTMs to provide a neural architecture that could learn to map input to output token sequences, which they used for machine translation. Vinyals et al. (2015) then used the same architecture for constituent parsing, by approaching it as a task akin to translating a foreign language with a sequence-to-sequence model. In their approach, the input sentence was transformed into a parenthesized tree, represented by a sequence of brackets. The sequence-to-sequence architecture was then adapted by other authors to predict other syntactic and semantic structures as a sequence-to-sequence problem, including tree-based (Li et al. 2018; Lin et al. 2022) and graph-based representations (Damonte and Monti 2021).

While both sequence-to-sequence and sequence labeling parsing approaches can be categorized as sequence-based models, because the output tree is represented as a linearized string, the main difference is that sequence labeling assigns exactly one label to each word, whereas in sequence-to-sequence models the length of the output sequence is arbitrary. Thus, the sequence-to-sequence approach is more flexible, allowing a wider range of linearizations and supporting complex semantic parsing formalisms that are out of reach for sequence labeling at the moment, such as AMR (Damonte and Monti 2021; Yu and Gildea 2022). However, the sequence labeling approach is simpler, more efficient (Anderson and Gómez-Rodríguez 2021), and easier to parallelize (Kitaev and Klein 2020) and integrate with other sequence-labeling tasks (Strzyz, Vilares, and Gómez-Rodríguez 2019a).

Dependency Parsing as Sequence Labeling. To our knowledge, the first implementation of a sequence-labeling encoding for dependency parsing in the neural era was in the work by Li et al. (2018). While the main focus of their paper was on sequence-to-sequence models, they tested a sequence labeling encoding for dependency parsing, which was based on computing relative distances between heads and dependents. However, the accuracy of this model was impractically low. The first approach that showed the viability of dependency parsing as sequence labeling, achieving reasonable accuracies, was shown later by Strzyz, Vilares, and Gómez-Rodríguez (2019b). They empirically showed, for a diverse set of languages, that choosing a compact mapping between words and labels was crucial for learnability with the BiLSTM architectures of the time. Since then, further improvements in machine learning (especially with pretrained language models) have made even naive encodings like that of Li et al. (2018) practically viable (Vacareanu et al. 2020). However, a good encoding design still matters, as highlighted by the variety of innovative encoding schemes introduced in recent years (Lacroix 2019; Strzyz, Vilares, and Gómez-Rodríguez 2020; Gómez-Rodríguez, Roca, and Vilares 2023; Amini, Liu, and Cotterell 2023; Ezquerro et al. 2025).

3. Constituent Parsing as Sequence Labeling

For a given sentence, we will define its sequence of words by $w = [w_1, w_2, ...w_{|w|}]$, where each word occurrence $w_i \in V$ (V is the full vocabulary of possible input words). Let $T_{|w|}$ be the set of constituent trees with $|w|$ leaf nodes, and without unary branches (in practice, unary branches can be handled by simple tree transformations, as will be explained below). We can define a set of labels L that allows us to encode each tree $t \in T_{|w|}$ with a sequence of labels $E_{|w|}(t) = [l_1, \ldots, l_{|w|}] \in L^{|w|}$, by means of an encoding function $E_{|w|} : T_{|w|} \rightarrow L^{|w|}$. To be useful for parsing, such a function must be **injective** (i.e., each tree must be mapped to a distinct sequence of labels) so that we can apply its inverse, $E^{-1}_{|w|}$, to a sequence of labels to recover the tree that originated it. We will call a function $E_{|w|}$ meeting this condition a **sequence labeling encoding** for constituent parsing.

The goal of casting constituent parsing in terms of sequence labeling therefore will be to predict a function $F_{|w|,\theta} : V^{|w|} \rightarrow L^{|w|}$, i.e., a function that maps a sequence of input words into labels, where V is the input word vocabulary, L is the label vocabulary, and θ is the set of parameters to be learned. With this, the parse tree for a sentence w can be obtained from $E^{-1}_{|w|}(F_{|w|,\theta}(w))$. Note, however, that the aforementioned injectivity condition only guarantees that $E^{-1}_{|w|}$ is defined on the image of $T_{|w|}$ through $E_{|w|}$, i.e., the

set of sequences of labels that result from encoding some tree. We call this set $W_{|w|} = \{E_{|w|}(t) \mid t \in T_{|w|}\}$ the set of **well-formed** label sequences for $E_{|w|}$. This restriction on the domain of $E^{-1}_{|w|}$ is problematic in theory, as a sequence-labeling system trained to predict $F_{|w|,\theta}$ can output ill-formed sequences (outside $W_{|w|}$). Ideally, we would like $E^{-1}_{|w|}$ to be defined on the whole $L^{|w|}$ (the set of all the possible sequences of $|w|$ labels from L), but this requires $E_{|w|}$ to be bijective, and none of the known encodings is bijective. In fact, bijectivity is impossible, as it would require $T_{|w|}$ and $L^{|w|}$ to have equal cardinality—but the number of possible label sequences is $|L|$ to the power of $|w|$, whereas the number of possible constituency trees scales with Catalan numbers, which cannot be expressed directly as an exponential function and grow strictly slower than $4^{|w|}$ (Stanley 2015). However, this theoretical issue is easily solved in practice via straightforward heuristics that convert invalid label sequences to valid ones, i.e., mapping from $L^{|w|}$ to $W_{|w|}$.

3.1 Unary Branch Handling

Our above definition of a sequence labeling encoding operates only on trees without unary branching. In practice, trees with unary branching can be handled in a uniform way across all encodings, by transforming them to remove unary chains. To our knowledge, all implementations of parsing as sequence labeling in the literature require this transformation. We define unary chain as a sequence of nodes $x_1, x_2, \ldots, x_k$ in a tree such that for each i from 1 to $k-1$, node x_i has only one child, which is x_{i+1}. We can differentiate between two kinds of unary chains:

- **Intermediate Unary Chains**: Those where x_k is not a terminal node (or a preterminal node, if they are used). To deal with them we will modify the tree, collapsing all their nodes into a new one whose label will be formed by concatenating the labels of the nodes in the chain, using a separator character (see Figure 2(a) and (b)). This is transparent for sequence labeling encodings, which will work with the collapsed tree nodes just as with any other.
- **Leaf Unary Chains**: Those where x_k is a terminal (or a preterminal node if they are used). While to deal with them we also apply the idea of collapsing the nodes into a single one, in this case the resulting node merges any nonterminal symbols in the chain together with a preterminal or terminal, directly associated with a given input word (see Figure 2(c) and (d)). This falls outside the scope of our definition of sequence labeling encodings for constituency parsing, which take the terminals and preterminals as fixed inputs, using them only as the leaf nodes on top of which the tree to be encoded is built. In other words, the encodings' goal is to predict the syntactic structure above the terminal or preterminal layer. For example, in Figure 2(c), the NT_i and w_i nodes are inputs and what we wish to encode is the tree above. But when we collapse the chain, obtaining Figure 2(d), one of the nonterminals in the tree (B) becomes integrated into the preterminal layer, so encodings as defined above will not represent it directly. However, it is easy to deal with this information: We extend encoding functions such that, for each input word, they output the labels of the nodes in the associated leaf

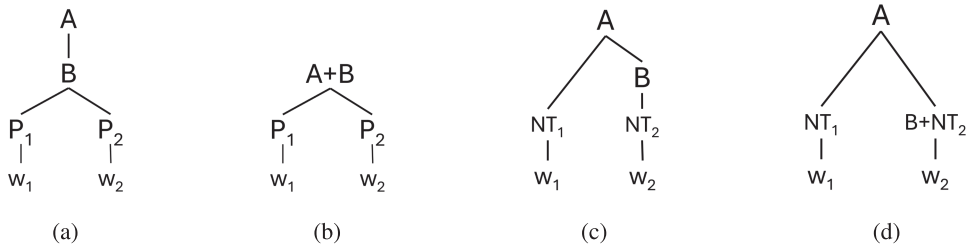


Figure 2
Example of an intermediate unary chain (a) and its collapsed form (b), as well as a leaf unary chain (c) and its collapsed form (d).

unary chain (e.g., *B* in the above example) in addition to the rest of the information in the syntactic label.

4. Tree Linearization Functions

We now describe different sequence labeling encodings, i.e., ways to define an injective encoding function $E_{|w|}$ that maps each tree in $T_{|w|}$ to a sequence of labels $[l_1, \ldots, l_{|w|}] \in L^{|w|}$. We include all the existing proposals in the literature we are aware of (as well as some proposals that were not initially conceived for sequence labeling, but are easily adaptable to this framework), along with a new alternative.

Broadly speaking, we can distinguish between two families of encodings: *depth-based encodings* and *linearized transition-based encodings*. While the former group relies on topological characteristics of the tree such as the height of nonterminal nodes, the latter creates the labels by mapping the actions in a transition-based system that generates the constituent tree to labels that are assigned to each word. While some encodings discussed in this article are conceptually based on multiple actions per word, our framework consistently packages this information into a single, composite label to maintain the strict one-to-one mapping that defines the sequence labeling paradigm.

Regardless of encoding family, we observe that all the encodings described in this article can be unified under a standard label format, according to how they represent the structure of the tree, its nonterminals, and leaf unary chains. Thus, to further facilitate homogeneous comparison and implementation, labels will always be 3-tuples of the form (s_i, c_i, u_i), where:

- s_i corresponds to the *structural* part of the encoding. Together, the set of s_i in the labels of a tree encode the shape of the tree, minus the node labels (i.e., nonterminal symbols) and leaf unary chains. The specific form of s_i is encoding-specific (e.g., it will be an integer value in depth-based encodings, but not in transition-based encodings).
- c_i encodes *constituent types*, i.e., the nonterminals present in the tree (excluding those in leaf unary chains). Depending on each specific encoding, each individual c_i can be a single nonterminal symbol or a list of them.
- u_i encodes the leaf *unary chains* mentioned in the previous section. This component behaves identically for all encodings: If there is a leaf unary

chain above word w_i, then the list of labels of the nodes involved in the chain is stored here. Otherwise, u_i is empty.

We now proceed to describe each of the families and their encodings.

4.1 Depth-based Encodings

The first family of encodings that we will explore is based on the syntactic distances between contiguous words in the sentence. In this family, the s_i component is an integer n_i which encodes depth information, related to the number of common ancestors between w_i and a contiguous word, either w_{i+1} or w_{i-1}, depending on the specific encoding and variant, as we will see in the next sections. On the other hand, the c_i field is defined in a common way across all encodings in this family: it represents the label of the deepest, or equivalently, the closest common ancestor between w_i and w_{i+1} (by default) or w_{i-1} (in the look-behind variants of the encodings, which will be explained in Section 4.1.3). Finally, u_i encodes the leaf unary chain in the standard way described previously.

4.1.1 Common Ancestor Encodings. Three distinct depth-based encoding functions have been described in the literature, having in common that the n_i field encodes the number of common ancestors between the words w_i and w_{i+1}, but differing in how it is specifically encoded:

Absolute Encoding. (`absolute`; Gómez-Rodríguez and Vilares 2018): This encoding stores in n_i the number of common ancestors between w_i and w_{i+1}. Figure 3 shows an illustrative example of a constituent tree linearized with this encoding, where, for example, $n_4 = 3$ because words w_4 and w_5 share three common ancestors (A, C, and D).

Relative Encoding. (`relative`; Gómez-Rodríguez and Vilares 2018): This encoding stores in n_i the difference between the number of common ancestors of w_i and w_{i+1} and the

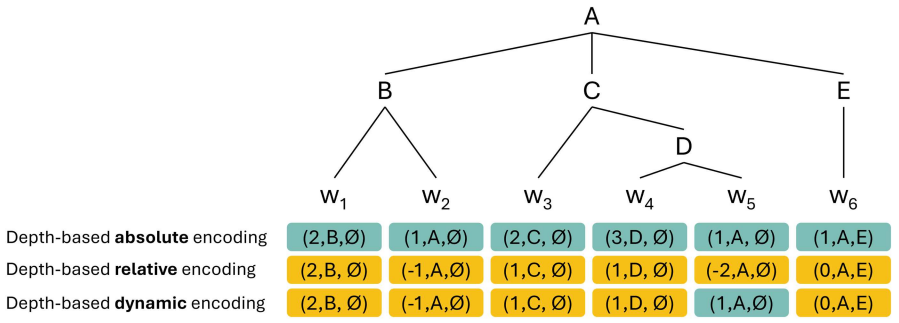


Figure 3
Example of a constituent tree and the labels obtained by encoding it with the `absolute`, `relative`, and `dynamic` encodings. Note that the n_i and c_i components for the last word are unneeded (as there is no next word w_7 with which to calculate a common ancestor) and unused in decoding. Our implementation fills these components by assuming a dummy last word linked to the root, yielding the values shown in the figure. Alternatively, one could set them to a null value.

number of common ancestors of w_{i-1} and w_i in the constituent tree. In other words, it represents the difference between the current and previous labels corresponding to the absolute encoding. For instance, Figure 3 also shows the same tree encoded with the relative encoding, where now $n_4 = 1$ because w_4 and w_5 have one more common ancestor (D) compared to w_3 and w_4. Also, n_4 now shares its value with n_3, and both indicate that a new constituent is opened for the word compared to the previous time step. In many cases, this helps compress multiple labels in the output vocabulary into a single label, which is generally easier for supervised models to learn.[2]

Dynamic Encoding. (`dynamic`; Vilares, Abdou, and Søgaard 2019): The advantage of `relative` over `absolute` is that it produces a reduced set of labels when opening or closing short constituents, which is a very common situation in real sentences. This relative approach makes most labels easier for the taggers to learn. For example, it is generally easier for a model to learn when to close a constituent than to track and maintain the entire depth of nested constituents at each time step. However, the relative encoding can be less effective when closing long constituents, which manifest as large negative n_i values that are scarce. Interestingly, the absolute encoding is more efficient in these cases: Typically, when closing many constituents, the word will belong to a constituent near the root of the tree, resulting in a small value for n_i under the absolute encoding. Considering this, the `dynamic` encoding combines `absolute` and `relative` in search for the best of both worlds. In this dynamic encoding scheme, a given word w_i is encoded using `relative` unless two conditions hold: that $n_i^{\text{rel}} \leq \alpha$ and $n_i^{\text{abs}} \leq \beta$, where α and β are adjustable threshold values and $n_i^{\text{rel}}, n_i^{\text{abs}}$ are the values of n_i under the relative and absolute encodings, respectively. If these two conditions are true, the word is encoded using `absolute` instead, i.e., the logic is that we prefer `absolute` in cases where `relative` would produce a large negative value and `absolute` produces a small positive one. We set $\alpha = -2$ and $\beta = 3$ based on Vilares, Abdou, and Søgaard's work. Figure 3 also contains the linearization of our example tree according to this dynamic encoding. Note that, for example, under this strategy $n_2 = -1$ comes from the relative encoding because the relative value $n_2^{\text{rel}} = -1$ is greater than α, regardless of the fact that $n_2^{\text{abs}} \leq \beta$. In contrast, $n_5 = 1$ comes from the absolute encoding, as the relative value $n_2^{\text{rel}} = -2$ is out of the desired range (i.e., $\leq \alpha$) whereas the absolute value $n_2^{\text{abs}} = 1$ is adequately small (i.e., $\leq \beta$).

Decoding. While we have `absolute`, `relative`, and `dynamic` depth-based encodings, the decoding process is common to all three of them, with the only difference being a preprocessing step to convert the relative and dynamic back into absolute scale. Once this is done, the decoding process for the absolute encoding starts by creating an empty root tree. Then, for each label l_i in the linearized tree, it descends along the rightmost branch to the level indicated by n_i (level 1 being the root), creating placeholder nodes with unspecified nonterminals along the way if needed. Once at the correct depth, it assigns the nonterminal to that node using the label's c_i field. After that, the algorithm attaches the word, together with its preterminal (if present) and leaf unary chain (if given in u_i) under the appropriate node. When $n_i > n_{i-1}$, corresponding to positive values in the relative encoding, this is always the newly created nonterminal node. When $n_i \leq n_{i-1}$, corresponding to negative or zero values in the relative encoding, it

2 This is not the case in the specific example we show, where the absolute encoding uses 3 distinct values of n_i and the relative encoding uses 5, but it is the case in many datasets, as will be seen in Table 1.

Table 1
Number of labels generated for each of the common ancestor encodings in the English Penn Treebank (PTB) and the SPMRL treebanks (German, Basque, French, Hebrew, Hungarian, Korean, Polish, and Swedish).

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
absolute	2342	4854	1392	2816	10710	2851	367	1349	4163	3427
relative	1854	7792	1832	2243	7068	3066	457	1143	3123	3175
dynamic	1599	7646	1857	2012	6297	3058	428	991	3041	2992

is the n_{i-1}th node in the tree's rightmost path, which (when $n_i < n_{i-1}$) will be further down the path from where the nonterminal was placed.

Table 1 shows the number of distinct labels that are generated when encoding all the trees from a sample of treebanks accessible to us. Specifically, we compute label counts for the constituent treebanks used in the experiments: the English Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993) and the multilingual SPMRL treebanks (Seddah et al. 2013).

The data shows that when comparing the `absolute` and `relative` encodings, neither is universally more compact than the other in terms of generated label set size. In most cases and on average, the `relative` encoding generates a smaller label space than `absolute`. This is especially visible in the Hebrew SPMRL treebank, where `relative` generates 30% fewer labels than `absolute`. This is the treebank with the largest average tree depth (see Table 8 for tree depth and other treebank statistics): In such a treebank, the absolute encoding is prone to generate large depth numbers, and thus the relative representation will compress the label space. However, there are also corpora where the relative encoding uses more labels, the most extreme case being German where it uses around 60% more, and which is also by far the shallowest treebank (Table 8). In this case, there are not many large absolute depths that the `relative` encoding can eliminate, but it will still add negative offsets.

In turn, the dynamic encoding is more compact than the other two in most datasets, since it tends to exploit the strengths of the relative encoding in the most favorable cases while avoiding potential sources of sparsity like large negative values when closing long constituents with deep nested structures.

4.1.2 Directional Depth-based Encodings. As a contribution of this work, we define two new depth-based encodings, which we call the left-descendant and right-descendant encodings. The left-descendant encoding is not entirely novel: In Gómez-Rodríguez and Vilares (2018), the authors proposed a simplified version of the relative encoding, which is only applicable for trees where all branchings have exactly k children. This stems from the observation that, in trees satisfying that condition, there is only one valid negative value of n_i in each given context. This is because one needs to add children to the lowest node in the rightmost tree path that does not yet have its k children before adding children further up the path. For this reason, one can map all negative values to a single label. Our left-descendant encoding is equivalent to that encoding for the case of binary trees ($k = 2$). However, we here formalize it in a much simpler way, present its natural counterpart (the right-descendant encoding, which is novel), and implement and evaluate both. This is especially relevant as the left-descendant encoding, which

had never been implemented before (the equivalent formulation outlined above was just described theoretically), obtains the best overall average accuracy among all the encodings considered in this article.

Left-descendant Encoding. (`l-desc`): In this encoding, which is only valid for binary or binarized trees, n_i is the number of ancestors of w_i that have w_i as their leftmost descendant. Equivalently, one can also describe n_i as:

- The number of common ancestors between w_i and w_{i+1} that have w_i as their leftmost descendant (in this case, requiring ancestors to be common with w_{i+1} is redundant, as all ancestors fulfilling the above condition are necessarily in common with w_{i+1}, but this definition serves to highlight the relation and differences of this encoding with respect to common ancestor encodings described before).
- The number of left child nodes one encounters before finding the first right child or the root in an upwards path from w_i,
- The number of open parentheses between w_{i-1} and w_i in the parenthetical representation of the tree.

Right-descendant Encoding. (`r-desc`): Analogously, in the case of the right-descendant encoding, n_i is the number of ancestors of w_i that have w_i as their rightmost descendant. Equivalently, one can also describe n_i as the number of common ancestors between w_{i-1} and w_i that have w_i as their rightmost descendant; or the number of right child nodes one encounters before finding the first left child or the root in an upwards path from w_i; or the number of closed parentheses between w_i and w_{i+1} in the parenthetical representation of the tree.

To generate the labels, one just needs to count left or right child links in upward paths from each word to the root.

Decoding. To decode the left-descendant encoding back to a tree, we use a left-to-right process. When the label for a word w_i has strictly positive n_i, we create a subtree with n_i left-branching tree levels, we set its lowest nonterminal to be c_i and link w_i as its left child, and then (if $i > 1$) attach the whole subtree as the right child of the lowest nonterminal that does not have one, taking advantage of the tree being binary. When $n_i = 0$, again using the fact that the tree must be binary, we directly link the current word w_i as right child of the lowest nonterminal that does not have one (see Figure 4, left-descendant encoding) and, if $i < |w|$, set c_i as the label for the lowest ancestor of that nonterminal that has not been assigned one (which by construction, will be the lowest ancestor that remains without a right child after w_i has been attached). For the right-descendant encoding, the process is analogous, but proceeds from right to left and swapping left with right for all intents.

An example of both encoding functions can be seen in Figure 4. Note for example how for the left-descendant encoding, $n_3 = 2$ because w_3 is a left child of a left child, and for the right-descendant encoding, $n_6 = 2$ because the leaf unary chain is a right child of a right child. An easily visible property of these encodings is that if n_i is positive for one of them, it is zero for the other, and vice versa, as a node is either a left or right child but not both.

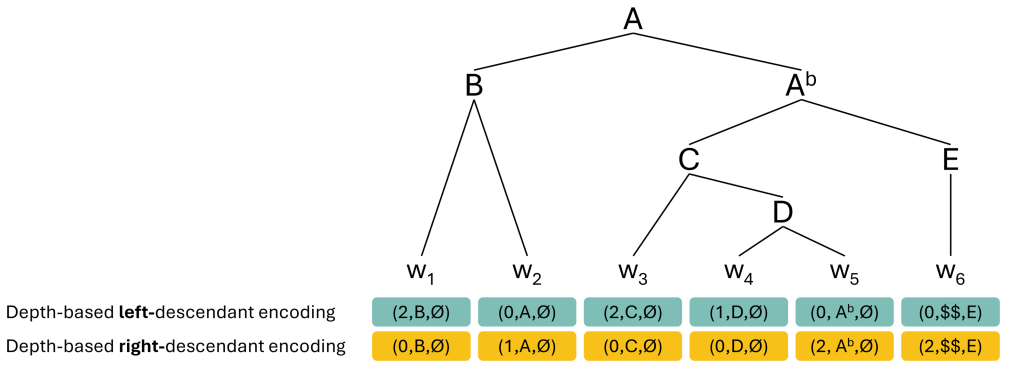


Figure 4
Example of a right-branching binarized constituent tree and the labels generated represented as a 3-tuple of the n_i, c_i, and u_i fields using both directional depth-based linearization functions.

Table 2
Number of labels generated for descendant encodings using both left (bl) and right (br) binarization for the English Penn Treebank (PTB) and the SPMRL treebanks (German, Basque, French, Hebrew, Hungarian, Korean, Polish, and Swedish).

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
r-desc$_{bl}$	2008	8270	1004	2917	4629	2818	363	464	3697	2908
r-desc$_{br}$	2186	8625	1166	2905	4789	3322	321	515	3642	3052
l-desc$_{bl}$	1420	9906	2100	2486	3985	4644	490	626	3482	3238
l-desc$_{br}$	915	6903	1095	1313	3107	2666	320	430	2829	2175

In Table 2 we show the number of labels generated for the directional depth-based encodings. Because these encodings require binarized trees, the table presents data on left and right binarized versions of the treebanks. Note that the effect of binarization (and its direction) on encodings that do not require it will be analyzed later in the article, with a discussion in § 4.1.3 and results in the experiments section (§ 6). The counts in Table 2 show that the suitability of each encoding in terms of label size is related to binarization direction, which is to be expected since right-branching trees will typically have longer chains of right children (which will generate larger values of n_i with the right-descendant encoding), and vice versa. Thus, we can see in the table that for right-binarized treebanks, the left-descendant encoding always produces a more compact label space, whereas for left-binarized treebanks, it is the right-descendant encoding that produces fewer labels in most cases and on average. Apart from binarization direction, the predominant branching direction of the underlying treebank (prior to binarization, see Table 8) also has some influence (albeit much milder), for the same reason. For example, the treebanks where differences are more stark in favor of the right-descendant encodings (comparing with the same binarization) are the Basque and Korean treebanks, which are predominantly left-branching (Table 8); whereas the left-descendant encoding behaves best in the English and French treebanks, which are predominantly right-branching. However, the Hebrew treebank goes against this trend (it is strongly left-branching, but the left-descendant encoding is more compact).

If we compare the number of labels of these two encodings with that of the common ancestor encodings in Table 1, we see that on average, the left-descendant encoding after right binarization is the most compact encoding considered so far, with considerable advantage. The rest of the encoding-binarization combinations, however, are not generally advantageous in this respect.

It is worth noting that, when applied to the same trees, one would generally expect `l-desc` and `r-desc` to generate fewer labels than `absolute` regardless of binarization direction (because the `l-desc` and `r-desc` encodings can also be seen as representing absolute counts of common ancestors, but they take into account only a subset of the common ancestors that `absolute` counts). However, in Tables 1 and 2, `absolute` is applied directly to trees as they are annotated in the treebanks, whereas for `l-desc` and `r-desc` the trees are binarized first, as this is a requirement for the encodings to work. Binarization makes trees deeper and introduces new nonterminals, thus producing more labels, hence the reduction in label set size is not universal. Comparisons where trees encoded with common ancestor encodings are *also* binarized will be provided later in the article.

4.1.3 Variants of Depth-based Encodings. We now discuss modifications that can be made to depth-based encodings. While minor from a conceptual point of view, these variants have not been tried in the literature and our experiments show that they can achieve substantial accuracy gains over the "default" versions of the encodings.

Look-behind Variants. Most of the depth-based encodings defined above have the characteristic that the label of a given word stores information that also involves the next word (look-ahead). This is most evident in the three depth-based common ancestor encodings introduced in §4.1.1 (where the label of w_i encodes the number of common ancestors between w_i and w_{i+1}) and arguable for the left-descendant encoding, under the interpretation of common ancestors with w_i as leftmost descendant. In order to avoid this and make the parsing closer to how humans understand language, we implemented a look-behind variant for each of the depth-based encodings where the label l_i is computed with information regarding w_i and w_{i-1} instead. This is equivalent to shifting the n_i and c_i components of each label one position to the right.

An example of this modification of the encodings can be seen in Figure 5. From now on, whenever we refer to the look-behind variant of an encoding X, we will denote it as X_{lb} (where X represents an encoding, like `absolute`, `relative`, etc.). Note that in the default look-ahead version, the last label is not actually useful for encoding depth information, as there is no subsequent word to count common ancestors with. In the look-behind variant, however, it is the first label that becomes redundant in this respect. Nonetheless, the $|w|$ labels must be computed in all cases, as they may be needed to represent leaf unary chains.

Binarization Variants. The three depth-based common ancestor encodings can work on arbitrary trees, without needing binarization. However, it is still interesting to test how they perform if trees are previously binarized, which is a common approach in other constituency parsing paradigms. Although this is not a conceptual contribution, this work includes, for the first time, experiments with this family of encodings to analyze the impact on the results of left- and right-branching binarizations, namely, referred to as X_{br} and X_{bl} for each encoding X. In the case of directional depth-based encodings, binarization is a requirement, but there is still the option to use different kinds of

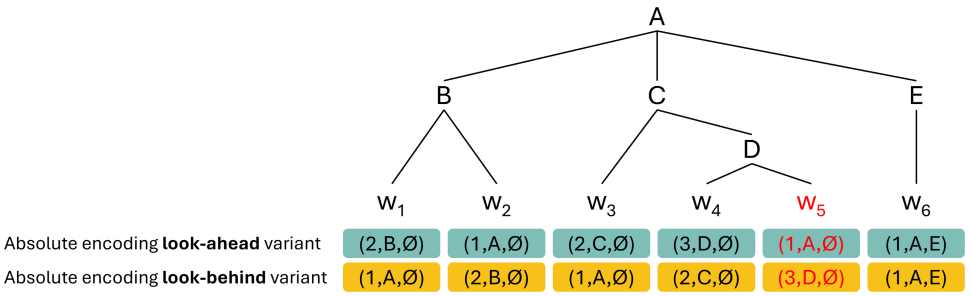


Figure 5
Example of a constituent tree encoded with depth-based absolute encoding and the look-behind variant of the depth-based absolute encoding. As seen in word w_5, for the default depth-based encoding we are computing the n_i and c_i fields with w_6, and therefore we obtain the node A and value 1, respectively, corresponding to the lowest common ancestor between w_5 and w_6. In contrast, for the look-behind variant we are obtaining the fields n_i and c_i using w_4, and therefore their values are D and 3, respectively.

binarizations so throughout the article we study the impact of choosing a left- or right-branching one, as we have already done in Table 2.

4.1.4 Disadvantages of Depth-based Encodings. From a theoretical perspective, all the encodings described above are unbounded, meaning that L (the set of possible labels) is not finite. This is because the range of possible values of n_i is not bounded by a constant. For example, in the case of `absolute`, n_i is the number of common ancestors between two contiguous words, and thus can range from 1 to the depth of the tree. The depth of a binary tree where unary branches have been collapsed (i.e., a full binary tree) ranges from $\Theta(log_2|w|)$ in the best case (complete tree) to $\Theta(|w|)$ in the worst case (left- or right-branching tree); so we can conclude that the size of the label set scales linearly with $|w|$ for the `absolute` encoding. While the other encodings attempt to address sparsity and produce more compact label sets in practical cases, they all ultimately work by representing lengths of paths in the tree, so analogous reasoning can be applied to show that all of them can produce $\Theta(|w|)$ labels in the worst case.

A consequence of unbounded encodings is that, even with a fairly large training set, it is possible to find new labels in the test set that were not previously seen in the training set. However, it has been shown previously in related setups that the number of unseen labels tends to be negligible in practice (Vilares and Gómez-Rodríguez 2020). Ultimately, this theoretical disadvantage has a minimal practical impact on neural models trained on these encodings, which still achieve competitive performance, as will be seen in the experiments.

4.2 Transition-based Linearizations

In addition to the initial depth-based encodings, transition-based algorithms for constituency parsing can be re-engineered to function within sequence labeling setups. Some of this work has remained purely theoretical (Gómez-Rodríguez, Strzyz, and Vilares 2020), mainly establishing sufficient conditions to map transition-based algorithms to sequence labeling, while other approaches have been implemented (Kitaev and Klein 2020). In any case, these conversions require that the processing of words occurs strictly from left to right, ensuring a consistent alignment between actions and

input words, which allows for direct mapping of subsequences of transitions to the corresponding input words. For this purpose, the presence of read transitions (which are typically shift transitions, although not exclusively, depending on the specific algorithm) is also required, as they are used to split transition sequences into $|w|$ subsequences, each used as a label associated to a word. Next, we present the main representatives of this family that have been put into practice and adapt other algorithms that had not previously been implemented in pure sequence labeling setups.

4.2.1 Tetra-tagging as Sequence Labeling. Tetra-tagging (`4tag`, Kitaev and Klein 2020) is a left-corner transition-based algorithm for constituency parsing. It is designed for binary trees where unary chains have been collapsed, so that every node has either 0 children (if it is a terminal node) or 2 children (if it is a nonterminal node). For a sentence of length $|w|$, such a tree will always have $|w| - 1$ nonterminal nodes, each positioned at a **fencepost**, i.e., the boundary between two consecutive words. Tetra-tagging works by assigning labels to all $|w|$ words and $|w| - 1$ fenceposts in a single left-to-right pass, using a total of $2|w| - 1$ labeling actions.

Thus, the algorithm can be viewed as word-synchronous parsing with $\Theta(|w|)$ theoretical inference time, which can function as a transition-based algorithm but is also straightforward to adapt to a sequence labeling setup. Kitaev and Klein's transition scheme is described next, assigning each word and fencepost a label. For clarity, we will denote the fencepost between words w_i and w_{i+1} by f_i.

- $\nearrow$: Transition associated with a word, indicating that its terminal is a left child of its parent node. This is a shift (i.e., read) transition.
- $\nwarrow$: Transition associated with a word, indicating that its terminal is a right child of its parent node. This is a shift (i.e., read) transition.
- $\nnearrow$: Transition associated with a fencepost f_i, indicating that the nonterminal symbol of the smallest span that crosses f_i (i.e., the lowest common ancestor of w_i and w_{i+1}) is a left child of its immediate parent.
- $\nnwarrow$: Transition associated with a fencepost f_i, indicating that the nonterminal symbol of the smallest span that crosses f_i (i.e., the lowest common ancestor of w_i and w_{i+1}) is a right child of its immediate parent.

By convention, the root node is considered to be a left child for the purpose of assigning a transition.

For any sentence we will have exactly $|w|$ occurrences of $\nearrow$ and $\nwarrow$ symbols, one per word, and each of them (except the last) will be followed by an occurrence of $\nnearrow$ or $\nnwarrow$, associated with the fencepost following said word. The $\nearrow$ and $\nwarrow$ transitions correspond, in the case of tetra-tagging, to the read transitions as defined by Gómez-Rodríguez, Strzyz, and Vilares (2020). Thus, we can create exactly $|w|$ subsequences of transitions, each of size 2 and with one word and one fencepost transition (except for that of the last word, which has size 1 as it needs no fencepost transition, although we could optionally include a dummy fencepost transition for uniformity with the previous labels). In this way, the parser can be trained as a tagging model. Figure 6 shows the sequence generated by this encoding, which we call tetra-tag in-order traversal encoding because the order in which it visits nonterminals (guided by lowest common ancestors between words) corresponds to an in-order traversal of the tree. The figure

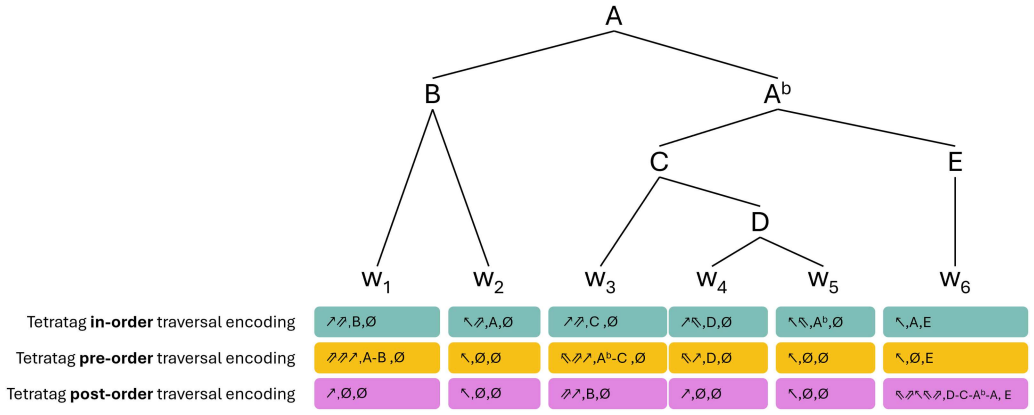


Figure 6
An example of a constituent tree and its corresponding tetra-tagging sequence labeling linearizations, as defined in Kitaev and Klein (2020) (in-order) and Amini and Cotterell (2022) (pre-order and post-order).

illustrates the subsequence derived from an input tree. As in any valid binary tree, the first action for w_1 is $\nearrow$ because w_1 is a terminal node and the left child of its nonterminal parent (as shown in the figure, labeled as a B constituent phrase). The second action, corresponding to the first fencepost f_1, is $\nrightarrow$ because B is the smallest span that crosses f_1 (or equivalently, the lowest common ancestor between the words separated by f_1), and B is the left child of its parent. The process then iterates for each subsequent word and fencepost, determining the suitable action at each timestep. Ultimately, this produces the sequence of labels shown in Figure 6 for the in-order version of the tetra-tagging algorithm. As explained above, the subsequence of actions for each pair (w_i, f_i) can be considered as a single label t_i assigned to each w_i, which can be predicted in a strict sequence labeling setup. For example, for w_1, $t_1 = \nearrow\nrightarrow$. With the sequence $(t_1, \ldots, t_{|w|})$, a complete unlabeled tree can be decoded, though without any assigned nonterminal symbols. This labeling scheme can easily be extended to our standard format of 3-tuples of the form $l_i = (s_i, c_i, u_i)$, where s_i is t_i, and c_i and u_i are defined as in the depth-based encodings and can be assigned deterministically during an in-order traversal of the tree.

It is worth noting that, when applied to trees without unary branches (thus, excluding the u_i component), this encoding is bounded, as the set of labels L is finite: t_i can take four distinct values (the four combinations of $\nearrow$ or $\nwarrow$ followed by $\nrightarrow$ and $\nleftarrow$), and c_i is bounded by the number of nonterminals in the annotation scheme.[3]

Decoding. The decoding algorithm reconstructs the tree using an in-order shift-reduce process, following the left-corner shift-reduce transition system detailed in Kitaev and Klein (2020). It processes the sequence of operators of each token (indicated in the s_i field of the label) one by one: a shift-right ($\nearrow$) builds a leaf (including its leaf unary chain, when applicable) and pushes it onto the stack, while a shift-left ($\nwarrow$) builds a leaf and immediately combines it with the tree on top of the stack, to fill an empty placeholder in said tree, which remains on the stack. Reduce operations ($\nrightarrow$ or $\nleftarrow$) create new

3 If we consider unary branches, no encoding can be bounded, as unary branches can have an arbitrary number of nonterminals that need to be encoded in the $|w|$ available labels.

Tetratag in-order		Tetratag pre-order		Tetratag post-order	
w1	↗	A	⇗	w1	↗
B	⇗	B	⇗	w2	↖
w2	↖	w1	↗	B	⇗
A	⇗	w2	↖	w3	↗
w3	↗	Ab	⇖	w4	↗
C	⇗	C	⇗	w5	↖
w4	↗	w3	↗	D	⇖
D	⇖	D	⇖	C	⇗
w5	↖	w4	↗	w6	↖
Ab	⇖	w5	↖	Ab	⇖
w6	↖	w6	↖	A	⇗

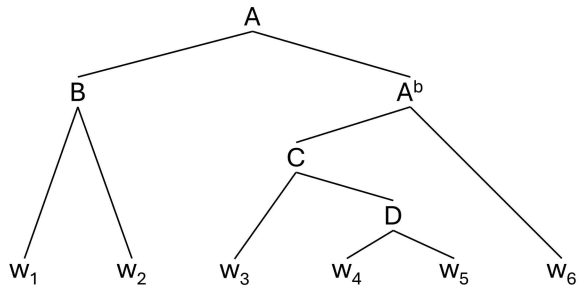


Figure 7
Example of the arrows generated per each node processed in the different traversals of tetra-tagging.

nonterminal nodes: In the left-child case (⇗), the new node takes the top of the stack as its left child and a placeholder as right child, the result being put back on the stack. In the right-child case (⇖), the algorithm pops the last subtree on the stack, attaches it as the left child of the new node, adds a placeholder as right child, and merges the resulting subtree filling a placeholder in the previous tree on the stack. After all tokens are processed, any remaining subtrees on the stack are combined until only one remains, which becomes the final constituent tree.

4.2.2 Tetra-tagging with Pre-order and Post-order Traversals. Building on the work of Kitaev and Klein (2020), Amini and Cotterell (2022) further explored how the tetra-tagging scheme could be applied to different mappings between words and subsequences of transitions. Specifically, while Kitaev and Klein use an in-order traversal of the tree to map words to labels (the sequence of transitions of the tetra-tagging transition system, split by shift transitions), Amini and Cotterell propose computing the sequence of transitions of other shift-reduce parsers whose actions can also be expressed in terms of ↗, ↖, ⇗, and ⇖, but where the difference is that the parsers operate based on pre-order or post-order traversals of the tree. Figure 7 shows an example of a tree and the order in which each transition is placed into the sequence of labels to be split depending on whether the traversal is in-order, pre-order, or post-order.

In this setup, sequences of shift actions (associated with ↖ and ↗ arrows) and reduce actions (associated with ⇖ and ⇗ arrows) are assigned to each label. For a given word w_i, the t_i and c_i fields encode the arrows and nonterminal labels that appear strictly after the *(i-1)*th read transition, up to and including the *i*th read transition.[4]

In contrast to in-order traversal, where the left-to-right ordering of nonterminals ensures that each label is assigned exactly two transitions (since there is a read transition every two labels) in pre-order and post-order traversals, the number of transitions (arrows) that can appear in a given label is unbounded. Consequently, these are considered unbounded encodings. This distinction has significant implications for label space

4 In the post-order traversal, non-read transitions occurring after the final read transition are appended to the last label, whereas in in-order and pre-order traversals, the final transition is always a read transition by construction.

complexity and transition behavior across different traversal strategies. To illustrate this, Figure 6 also presents the labels obtained for the pre-order and post-order traversals on an example tree, highlighting the structural differences among these encoding variants with respect to the original in-order encoding.

In addition, in Figure 8, we show the number of actions that each word encodes for each traversal in the test set of the English Penn Treebank. As we can see, while in pre-order traversal, the encoding tends to stack actions at the beginning of the sentence, and in post-order traversal, it tends to stack them at the end of the sentence, in-order traversal maintains a constant rate of two actions per word, independent of the word's position within the sentence. Similar conclusions can be drawn regarding nonterminal symbols: In-order tetra-tagging encodes exactly one nonterminal per word, whereas pre-order and post-order traversals encode them at a nonconstant rate, stacking more near the beginning or end of the sentence, respectively.

This is related to the concepts of alignment and deviation as defined by Amini and Cotterell (2022). They show that, out of the three traversals, only the in-order one makes it possible to satisfy two properties simultaneously: paired alignment (each word being assigned the same number of actions, i.e., two) and zero deviation (with deviation being defined as the distance between each word w_i and the word whose tag contains the shift action that pushes w_i to the stack). In a transition-based implementation, one

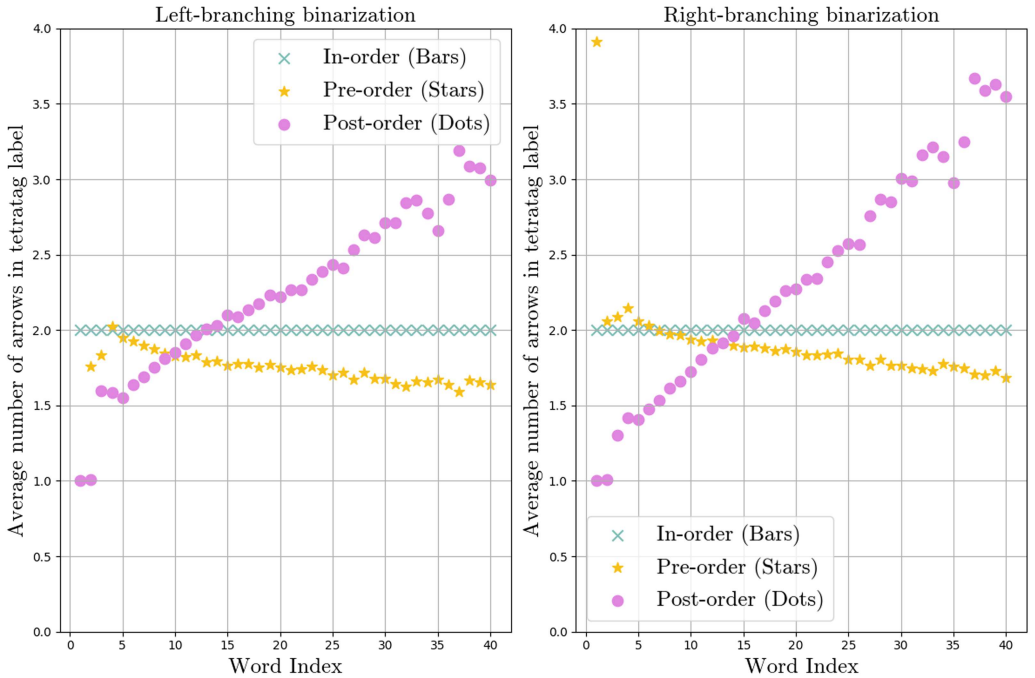


Figure 8
Average number of actions encoded per label in the different tetra-tagging variants, plotted with respect to word/label index (i.e., position in the sentence). The left figure illustrates the case of left-branching binarized trees, while the right figure represents right-branching binarized trees. We can see how, while the in-order traversal tetra-tagging always stores exactly two actions per word, the pre-order version stores more actions in the labels of the initial words of the sentence, and the post-order version in those of the final words. Similar trends arise if we look at encoded nonterminals instead of actions.

can enforce paired alignment but then will incur deviation in the pre- and post-order traversals, which Amini and Cotterell (2022) showed to negatively impact performance. As we are implementing these encodings into a sequence labeling paradigm instead, the situation is the opposite: Because each label is directly associated with a word, it is more natural to enforce zero deviation (by doing shift alignment, i.e., splitting label sequences using read transitions). However, the consequence is that we forgo paired alignment in the pre- and post-order traversals, which, as we will see, will also negatively impact performance. For completeness, we did also implement alternative versions of pre-order and post-order tetra-tagging under paired alignment (i.e., assigning two actions per word, and accepting deviation). However, in our sequence-labeling setup this variant consistently underperformed compared to the zero-deviation alternative described above. We therefore only report results for the zero-deviation versions in the tables.

Decoding. The decoding processes for the pre-order and post-order variants of tetra-tagging are similar to that of the in-order version, using the same primitives, but they build the tree in a different order. In the pre-order variant, the decoding algorithm builds the tree top–down. The stack holds a path of nodes from the root, where each node is awaiting its children to be built. For each token, the sequence of operators in its label is processed sequentially. A $\nearrow\!\!\!\!\!+$ operator creates a new nonterminal node with two empty placeholders for children and pushes it to the stack; if the stack is not empty, this node also becomes the left child of the previous top element. A shift-right $\nearrow$ operator fills the leftmost placeholder of the topmost stack node with a leaf (the word, any preterminal, and unary chain if applicable). A shift-left $\nwarrow$ operator similarly attaches a leaf to the rightmost placeholder of the top node, closing it (as it has found its two children) and popping it from the stack. Finally, a $\nwarrow\!\!\!\!\!+$ operator creates a new nonterminal node and attaches it to the rightmost placeholder of the topmost stack node, which is removed from the stack (as it has found its two children) and replaced with the new node.

For the post-order variant, we rebuild the tree in a bottom–up fashion. The stack will contain subtrees, which will be marked according to whether we expect them to be linked to a larger subtree either as left or as right children. The $\nearrow$ and $\nwarrow$ operations create a new leaf (with a word, any preterminal and unary chain, if present) and push it to the stack, expecting to be linked as left or right child, respectively. The $\nearrow\!\!\!\!\!+$ and $\nwarrow\!\!\!\!\!+$ operations create a nonterminal node and attach the two topmost subtrees on the stack as its left and right children (if the transition sequence is well-formed, the second topmost should be expecting to be linked as left child, and the topmost as right child). Then, these two stack elements are replaced with the newly-built subtree, which will be marked as expecting to be a left or right child if the operation was $\nearrow\!\!\!\!\!+$ or $\nwarrow\!\!\!\!\!+$, respectively.

Table 3 shows the number of labels generated by the different variants of the tetra-tagging encodings across treebanks and binarization directions. In the table and from now on, we denote the variants with subindices in (in-order), pr (pre-order), and po (post-order). As expected, the in-order variant has, by far, the most compact label space among tetra-tagging encodings, a result consistent across all treebanks. This is easily explained by paired alignment, with each label containing exactly two arrows in the t_i component and one nonterminal in the c_i component. In the other variants, the lack of paired alignment increases label diversity, producing the largest label counts among the encodings covered so far.

4.2.3 Attach-juxtapose Parsing as Sequence Labeling. The attach-juxtapose algorithm, as proposed by Yang and Deng (2020), is a strongly incremental decoder designed for constituent trees. In the original implementation, this decoder is applied on top of

Table 3
Number of labels generated for different tetra-tagging (4tag) encodings across various treebanks: the English Penn Treebank (PTB) and the SPMRL treebanks (German, Basque, French, Hebrew, Hungarian, Korean, Polish, and Swedish). We consider both right and left binarization (denoted by the br and bl subscripts), as well as all three available traversals.

Tetra-tag Variant	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
$4tag_{blin}$	1022	7690	1660	1590	5397	3134	217	853	3239	2756
$4tag_{brin}$	1037	7157	1389	1435	5201	2855	172	791	3220	2584
$4tag_{blpr}$	4563	16884	8360	3685	5627	10982	6557	1497	4066	6913
$4tag_{brpr}$	2329	14629	5655	3054	5635	6554	3849	1036	4075	5202
$4tag_{blpo}$	25078	23525	3524	23607	15699	6149	739	3457	8082	12207
$4tag_{brpo}$	40731	28035	4342	27877	14112	8434	1328	4834	7938	15292

a graph convolutional network that encodes partially built (n-ary) trees. Next, we summarize the proposed transition-based encoding (att-jxt) and describe how it can naturally be cast in terms of a sequence labeling encoding, which is the version we will consider within the scope of this work. In the attach-juxtapose transition-based algorithm, *attach* and *juxtapose* are the only two possible actions at each timestep. Both actions add a word w_i to a partial tree but modify said partial tree differently. We will denote the partial tree that has been built after the action at timestep i as $\mathcal{T}_i$. In turn, $\mathcal{R}(\mathcal{T}_i)$ will denote the rightmost spine of $\mathcal{T}_i$, i.e., the path that starts at the root node and iteratively descends to the rightmost leaf.

Attach (target node, parent label). Given a partially built tree $\mathcal{T}_{i-1}$ computed at timestep $i-1$, the *attach* action adds w_i as a right-descendant of a target node in $\mathcal{R}(\mathcal{T}_{i-1})$. To accomplish this, a nonterminal node, *prt*, is created as the parent of w_i and assigned the specified parent label. The subtree rooted at *prt* is then attached as the rightmost child of the target node, which is defined as the node at level l of $\mathcal{R}(\mathcal{T}_{i-1})$, with level zero representing the root of the tree. Figure 9 provides an example of this *attach* action. It is also possible to not provide a parent label, with the effect that the word (in the example in the figure, with preterminal P_3) is attached directly as the rightmost child of the target node instead.

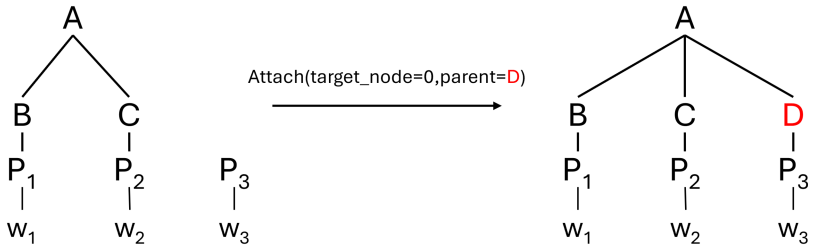


Figure 9
Example of executing the attach action for the subtree formed by the word w_3, assuming that the preterminal P_3 has already been computed with a parent labeled D. The new partial tree is attached as the rightmost child of the node at level zero of the tree's rightmost spine.

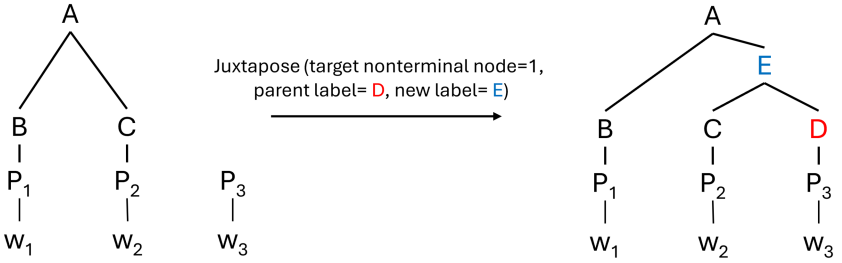


Figure 10
Example of executing the juxtapose action for the subtree formed by the word w_3, assuming the preterminal P_3 is already computed. The operation places a new nonterminal node, E, and positions a subtree labeled D with w_3 as the rightmost child of E.

Juxtapose (target nonterminal node, parent label, new label). Given $\mathcal{T}_{i-1}$, the *juxtapose* operation replaces a target nonterminal node, *tgt*, within $\mathcal{R}(\mathcal{T}_i)$ with a new nonterminal node, *new*. In this configuration, *tgt* becomes the left child of *new*, while a new subtree—consisting of *prt* and w_i—is attached as the rightmost child of *new*. Similar to the attach transition, the new nonterminal node is located as the rightmost child at the l-th level of $\mathcal{R}(\mathcal{T}_i)$, starting from the root. Figure 10 illustrates this *juxtapose* action.

Interestingly, this transition-based algorithm exhibits properties that make it easily adaptable to a sequence labeling setup. Specifically, for a sentence of length n, the algorithm produces exactly n transitions. Since each transition simultaneously adds a word to the tree—effectively making every transition a "read" transition—the pairing between words and transitions is straightforward and direct. This enables learning the parsing algorithm within a strict sequence labeling framework, introducing a new transition-based encoding to the existing family of in-order, pre-order, and post-order tetra-tagging variants.

To structure the labels in our standard format, we store in s_i the transition (Attach or Juxtapose) together with its integer parameter (target node), while c_i encodes the associated nonterminal labels: the parent label (or empty) for Attach transitions, and the parent and new labels for Juxtapose transitions. The u_i field is the same as in previous encodings. An example of this encoding function is shown in Figure 11.

Decoding. For this algorithm, the decoder reconstructs the tree using the explicit actions encoded in each label. It begins with an empty root and, for every token, retrieves from the s_i field the action type (attach or juxtapose) and target tree level, and from the c_i field the optional parent or new labels. A terminal or preterminal node is then built for the word (including the corresponding leaf unary chain, if present). The decoder descends the current tree's rightmost spine until reaching the target depth or a preterminal, and then applies the action: attach adds the new subtree as the rightmost child of the target node, optionally wrapped in a parent node with the specified parent label, while juxtapose creates a new parent node with the target subtree and the new one as children under the specified label, replacing the target in the tree.

Table 4 shows the number of distinct labels of the attach-juxtapose encoding on different treebanks, both unbinarized and with left and right binarization. The results show that this is a reasonably compact encoding when applied to unbinarized trees, but label size greatly expands with binarization, especially right binarization. This can owe to the fact that long right-branching trees will have long right spines (generating

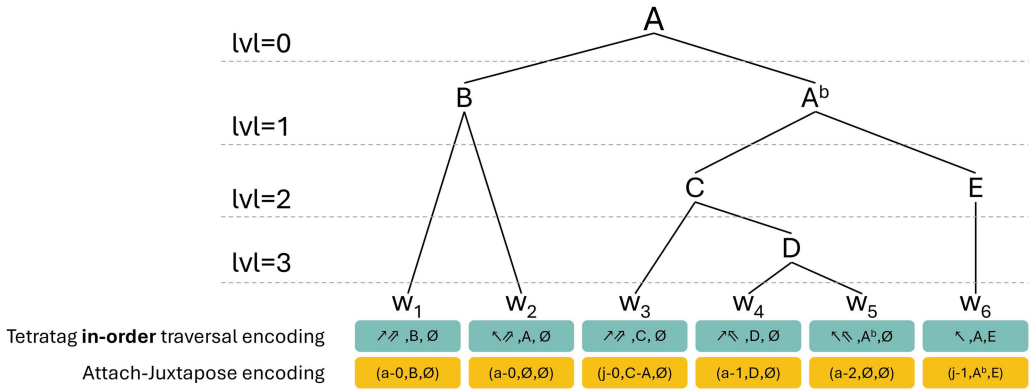


Figure 11
Example of a constituent tree with labels generated using the attach-juxtapose encoding, compared to in-order tetra-tagging. Labels are structured as a 3-tuple $l_i = (s_i, c_i, u_i)$, where the s_i field indicates the action type (a for Attach or j for Juxtapose) together with the numerical parameter indicating the target node's level, while c_i contains the associated nonterminals, if any.

Table 4
Number of labels generated for Attach-Juxtapose encoding using both left and right binarization (bl and br subscripts) for the English Penn Treebank (PTB) and the SPMRL treebanks (German, Basque, French, Hebrew, Hungarian, Korean, Polish, and Swedish).

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
`att-jxt`	2569	5794	1554	3893	12150	2611	287	1795	3464	3791
`att-jxt`$_{bl}$	6474	17219	3376	13024	17133	5342	521	3021	8015	8236
`att-jxt`$_{br}$	5942	27518	5643	15569	23246	12488	566	3969	9277	11580

wider ranges of level numbers in the t_i component) and require more juxtapose actions, which take a larger toll on label set size than attach actions (each juxtapose action takes two nonterminal parameters, so their potential amount is quadratic on the number of nonterminals).

5. Sequence Labeling Neural Architecture

Sequence labeling is a structured prediction task in which, given a sequence $\vec{w} = [w_1, w_2, \ldots, w_n]$ of n input tokens, the model outputs a corresponding sequence of n labels, with each token receiving exactly one label. Supervised learning is the dominant approach for solving these tasks. Although generative LLMs can be prompted to generate specific outputs, fine-tuning with a carefully selected set of labels achieves the best trade-off between effectiveness and efficiency in training supervised models. These labels are often ad hoc, domain-specific, and can be numerous. This approach is particularly relevant to our work, as evidence suggests that large language models are not well-suited to predict structured outputs associated with parsing tasks (Lin et al. 2023; Ezquerro, Gómez-Rodríguez, and Vilares 2025), even when evaluated on official test sets, where they may rely on memorization. Consequently, our methodology, building

on prior research in parsing-as-tagging, adopts this standard supervised approach with an encoder-decoder architecture.

In neural networks for sequence labeling, an encoder maps tokens to contextualized representations by processing the entire sentence. Various sequence models can serve as the encoder, including convolutional, recurrent, or Transformer networks. In this work, we use the Transformer architecture, leveraging pretrained models optimized for masked language modeling, which have consistently demonstrated superior performance for parsing-as-tagging tasks (Vilares et al. 2020; Vacareanu et al. 2020). For the decoder, we follow the common approach of using a simple feed-forward network to generate the output label for each token based on its hidden representation from the encoder.

In this context, a relevant implementation detail arises from the mismatch between the input units of our parsing encodings, which assign one label per word, and the input required by Transformer encoders, which operate on subword tokens. In our implementation, the encoder processes subwords, but labels are predicted at the word level: When a word is split into multiple tokens, we take the hidden representation of its first subword and use it to predict the word-level label. This ensures consistency between the model's predictions and the gold annotations.

Formally, let $\vec{t} = [t_1, t_2, \ldots, t_m]$ represent the sequence of m subword tokens obtained from tokenizing an input sequence of n words, where each word corresponds to one or more subword tokens. Each subword t_i is embedded into a vector space through an embedding layer. The neural encoder processes the embedded sequence to produce a sequence of contextualized hidden states $\vec{h} = [\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_m]$, where $\vec{h}_i \in \mathbb{R}^d$ captures contextual information around the i-th token. In our case, the encoder is a Transformer model, which computes each $\vec{h}_i$ through self-attention mechanisms, allowing each token to attend to all other tokens in the sequence. Particularly, for this work, we use XLM-RoBERTa as our encoder (Conneau 2019).

To produce the final output, we aggregate subword representations at the word level by selecting the hidden state of the first subword of each word, as mentioned above. The decoder is then applied to these n selected representations to generate an output label $\hat{y}_i$ for each word, where $\hat{y}_i = \text{argmax}(f(\vec{h_i}^{(1)}))$, with $\vec{h_i}^{(1)}$ denoting the hidden state of the first subword of word w_i. This defines the mapping $\vec{w} \rightarrow \vec{\hat{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n]$.

The model parameters are optimized via supervised learning to minimize a loss function, a categorical cross-entropy, given by $\mathcal{L} = -\sum_{j=1}^{L} y_{i,j} \log(\hat{y}_{i,j})$ for each token i, where L is the total number of labels for a given encoding, $y_{i,j}$ is the ground truth for class j, and $\hat{y}_{i,j}$ is the predicted probability for class j.

Note that for all constituent tree linearizations, each label associated with a word is represented as a triplet where each component $y_i^{(k)}$ (for $k = 1, 2, 3$) represents a distinct sub-label corresponding to a specific aspect of the syntactic label. To address the multi-component structure of these labels, we rely on a hard-sharing multi-task learning setup, in which each component $y_i^{(k)}$ is treated as a separate yet related task, creating three parallel sub-tasks. In particular, for each sub-task k, a dedicated task-specific decoder is applied to $\vec{h}_i$ to predict the corresponding sub-label $y_i^{(k)}$. This structure allows the model to jointly learn both shared and task-specific features, optimizing the overall objective by minimizing a combined loss function $\mathcal{L} = \sum_{k=1}^{3} \mathcal{L}^{(k)}$, where $\mathcal{L}^{(k)}$ represents the loss for each sub-task. For this, we rely on the MaChAmp framework (van der Goot et al. 2021), a toolkit for training various types of NLP tasks—including sequence labeling—in a black-box fashion.

6. Experiments

We now move to the empirical evaluation of all encodings under a homogeneous setup, introducing the datasets used, the metrics we rely on, and the experiments conducted.

Datasets. We will evaluate our encodings using standard treebanks. For English, we rely on the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). Additionally, we evaluate on morphologically rich languages using the SPMRL treebanks (Seddah et al. 2013): German (de), Basque (eu), French (fr), Hebrew (he), Hungarian (hu), Korean (ko), Polish (pl), and Swedish (sv).[5] Table 8 presents various statistics for the original treebanks to help us study potential influences on encoding performance.

Metrics. For parsing performance, as is standard, we report the labeled bracketing F1-score, which measures accuracy over bracketed spans. For evaluation on the Penn Treebank, we use the `evalb`[6] tool with the `COLLINS.prm` file, and for the SPMRL dataset, we use `evalb_spmrl` with the parameter file `SPMRL.prm`. F1-scores are reported with respect to the unbinarized treebank, i.e., in the cases where binarization is used for parsing, trees are unbinarized before evaluation.

Postprocessing. As mentioned in Section 3, a common theoretical limitation of every sequence-labeling encoding is lack of bijectivity. In practical terms, this implies that not all possible label sequences encode a valid tree. In practice, this is usually tackled by simple postprocessing heuristics to obtain valid trees from ill-formed label sequences. In this respect, for comparison to be fair, we ensure that all encodings follow the same heuristics, which are the following:

- If the label sequence assigns several conflicting nonterminal labels to the same node, the majority label is picked, or the first one if there is a tie.
- If the nonterminal label of a node is not specified, the node is removed from the tree (by linking its children directly to its parent).
- In depth-based encodings, indexes out of range are changed to the closest legal index. For example, a -5 in a relative encoding is interpreted as a -4 if there are only 4 tree levels to ascend.
- In transition-based encodings, if the label sequence specifies an illegal transition (e.g., an attach or juxtapose transition whose parameters do not point to a valid node), the transition is skipped and the associated word is instead attached at the current tree level. Any words that remain unattached to the tree at the end of the process are attached to the rightmost lowest nonterminal.
- If any artificial nodes associated with binarization remain after unbinarization, they are removed by linking their children directly to their parent.

5 We exclude Arabic as we do not have access to the proprietary license.
6 `https://nlp.cs.nyu.edu/evalb/`.

Table 5
Percentage of trees that required heuristic postprocessing for each of the main encodings during the decoding process of the predicted test labels.

Encoding	PTB	SPMRL								Average$_{std}$
		de	eu	fr	he	hu	ko	pl	sv	
absolute	13.08	7.70	7.82	10.66	60.47	7.73	20.16	9.49	10.33	$16.38_{16.02}$
relative	5.84	4.54	6.45	11.57	30.59	5.45	15.74	13.50	9.61	$11.48_{7.69}$
dynamic	6.04	3.56	7.08	7.01	26.40	5.05	18.06	4.14	13.36	$10.08_{7.29}$
l-desc$_{br}$	7.70	6.06	17.55	9.72	22.49	12.49	14.56	5.47	14.54	$12.29_{5.32}$
r-desc$_{br}$	25.79	18.50	20.40	9.23	49.66	35.28	11.59	14.96	45.80	$25.69_{13.89}$
4tag$_{brin}$	4.97	2.88	4.33	9.06	14.66	6.74	10.13	0.61	11.11	$7.17_{4.20}$
att-jxt	8.43	7.75	5.41	9.78	8.36	8.90	19.97	9.51	10.27	$9.82_{3.83}$

Table 5 shows the percentage of trees in each treebank for which our model produced an ill-formed label sequence in the experiments, thus requiring heuristic postprocessing to map them back into a valid constituent structure. For readability reasons, we show a single variant per encoding. Overall, the majority of generated label sequences are valid, with the percentages of ill-formed sequences typically falling into the 5–15% range. This means that although requiring postprocessing is not uncommon, an ample majority of sentences do not need it, with the model generating a fully well-formed sequence of labels by itself. It is worth considering that these statistics refer to whole sentences, but in most cases where heuristics are used, they only apply to one or a few labels in a long sentence, leaving most unchanged. Thus, the overwhelming majority of labels are not touched by postprocessing.

The proportion of ill-formed outputs varies widely across encodings. The right-descendant and absolute encoding require postprocessing most often, probably due to the sparsity of their label sets and the consequent difficulty of keeping track of large depths. This may lead the encodings to leave nodes without a nonterminal (if a too large depth is predicted) or to assign conflicting nonterminals (if the depth is too small). On the other side, in-order tetra-tagging is the most reliable, producing ill-formed sequences for just 7.17% of trees on average. This is to be expected, as it is a bounded encoding using a small, fixed set of action symbols which gives rise to only four possible values of the label component s_i, so the space of possible erroneous sequences that do not correspond to valid trees is smaller.

In terms of languages, the Hebrew dataset (which has the deepest trees among the datasets tested, as well as rather long sentences; see Table 8) requires the most postprocessing.

For a more fine-grained analysis of the impact of postprocessing heuristics, Table 6 compares label accuracy (i.e., percentage of the predicted labels that coincide with those of the correct encoding of the gold tree) before and after the postprocessing step. The differences are generally of around one percentage point, much lower than the proportion of trees needing heuristic postprocessing. This is the case even in datasets like Hebrew, where the proportion of ill-formed label sequences (Table 5) was atypically high. This confirms that the decoding heuristics typically affect only a small portion of the constituent tree, being localized in one or a few labels. Thus, even for datasets where a relatively high proportion of label sequences requires some postprocessing, the overall impact is not high. As expected, encodings for which the model produces ill-formed output less often (Table 5), like 4tag or att-jxt, show smaller label accuracy differences

Table 6
Increase in label accuracy after heuristic postprocessing (i.e., delta after minus before postprocessing) for each of the main encodings and datasets.

Encoding	PTB	de	eu	fr	he	hu	ko	pl	sv	Δ_{Average}
`absolute`	0.98	1.00	0.87	1.27	0.89	1.24	1.33	0.73	0.84	1.02
`relative`	1.36	1.16	1.14	1.84	1.40	1.27	1.44	0.75	0.93	1.25
`dynamic`	1.24	1.25	0.96	1.70	0.95	1.18	1.37	0.57	1.57	1.20
$\texttt{l-desc}_{\text{br}}$	1.06	0.90	1.03	1.75	0.91	1.12	1.07	0.38	0.65	0.99
$\texttt{r-desc}_{\text{br}}$	2.25	1.03	1.12	1.57	0.98	0.88	1.52	1.07	0.57	1.22
$\texttt{4tag}_{\text{br in}}$	1.24	0.42	0.88	0.97	1.06	0.88	0.99	0.22	0.33	0.78
`att-jxt`	1.09	0.72	0.58	1.03	0.97	0.59	1.21	0.32	1.63	0.90

in Table 6, supporting the idea that they are more robust during decoding. Overall, these results confirm that while the need for postprocessing to ensure generation of valid trees is relatively common, its impact on accuracy is generally limited and consistent across the different tested languages.

6.1 Label Set Sizes and Distributions

Before addressing the result of parsers trained with the different encodings, we first show the number of labels generated for each treebank and encoding on Table 7. This puts together the data gathered in Tables 1 to 4, facilitating comparisons across encoding groups; and also provides more fine-grained information: number of distinct values of the individual fields s_i and c_i (we exclude u_i as it is encoding-independent, so it takes a constant value for each treebank), and data for all binarization variants (and unbinarized treebanks, where applicable) as well as for the look-behind (lb) variants of each encoding.

Apart from the conclusions drawn from the comparisons within each encoding group, covered in previous sections, we highlight the following relevant insights from the big picture in Table 7:

- The left-descendant encoding with right binarization ($\texttt{l-desc}_{\text{br}}$) is the most compact encoding on average (2,175 labels), in spite of being unbounded in theory. The next most compact is the dynamic encoding without binarization (`dynamic`) but at considerable distance (2,549 labels), closely followed by in-order tetra-tagging with right binarization ($\texttt{4tag}_{\text{br in}}$), with 2,584.
- The number of values of the component c_i confirms that the negative impact of binarization on label set size is largest in encodings where one label can contain more than one nonterminal (making the size quadratic in the number of nonterminals), i.e., pre-order and post-order tetra-tagging and attach-juxtapose.
- When comparing binarizations, right binarization produces fewer labels than left binarization for most encodings, the exceptions being `absolute`, `r-desc`, and `att-jxt`.
- The look-behind variant tends to increase label set size.

Table 7
Number of labels generated for each implemented encoding and its variations for the English Penn Treebank and the SPMRL datasets, indicated as a^c_b where a is the total number of unique labels and b, c are the number of unique values of the fields s_i and c_i for each label, respectively.

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
absolute	2342^{109}_{35}	4854^{2450}_{11}	1392^{45}_{11}	2816^{203}_{32}	10710^{274}_{29}	2851^{185}_{14}	367^{16}_{16}	1349^{50}_{19}	4163^{320}_{21}	3427^{406}_{21}
absolute$_{bl}$	4163^{187}_{54}	19550^{4170}_{65}	4980^{86}_{39}	7883^{364}_{68}	17687^{455}_{88}	10637^{327}_{74}	713^{31}_{30}	2358^{82}_{25}	7450^{541}_{39}	8380^{694}_{54}
absolute$_{br}$	4702^{187}_{56}	20359^{4170}_{63}	4793^{86}_{35}	9454^{364}_{74}	19951^{455}_{78}	12686^{327}_{67}	527^{31}_{29}	2763^{82}_{29}	8707^{541}_{60}	9327^{694}_{54}
absolute$_{lb}$	2820^{109}_{35}	4807^{2450}_{11}	2354^{45}_{11}	3769^{203}_{32}	14449^{274}_{29}	4381^{185}_{14}	468^{16}_{16}	2600^{50}_{19}	5124^{320}_{21}	4530^{406}_{21}
absolute$_{lbbl}$	4828^{187}_{54}	19261^{4170}_{65}	7067^{86}_{39}	9535^{364}_{68}	23763^{455}_{88}	14090^{327}_{74}	895^{31}_{30}	4382^{82}_{25}	9014^{541}_{39}	10315^{694}_{54}
absolute$_{lbbr}$	5296^{187}_{56}	19925^{4170}_{63}	6342^{86}_{35}	11373^{364}_{74}	25562^{455}_{78}	15318^{327}_{67}	762^{31}_{29}	4551^{82}_{29}	9662^{541}_{60}	10977^{694}_{54}
relative	1854^{109}_{40}	7792^{2450}_{19}	1832^{45}_{17}	2243^{203}_{29}	7068^{274}_{34}	3066^{185}_{20}	457^{16}_{25}	1143^{50}_{27}	3123^{320}_{25}	3175^{406}_{26}
relative$_{bl}$	3243^{187}_{62}	12812^{4170}_{65}	3971^{86}_{36}	4695^{364}_{76}	10209^{455}_{60}	6227^{327}_{55}	747^{31}_{32}	1815^{82}_{39}	4761^{541}_{40}	5386^{694}_{52}
relative$_{br}$	2846^{187}_{50}	10132^{4170}_{59}	2595^{86}_{38}	3571^{364}_{57}	9488^{455}_{50}	4783^{327}_{53}	548^{31}_{38}	1682^{82}_{35}	4226^{541}_{40}	4430^{694}_{47}
relative$_{lb}$	1608^{109}_{40}	7549^{2450}_{19}	2351^{45}_{17}	2210^{203}_{29}	8771^{274}_{34}	3681^{185}_{20}	579^{16}_{25}	1824^{50}_{27}	3557^{320}_{25}	3570^{406}_{26}
relative$_{lbbl}$	2965^{187}_{62}	12478^{4170}_{65}	4307^{86}_{36}	4828^{364}_{76}	12598^{455}_{60}	6735^{327}_{55}	918^{31}_{32}	2741^{82}_{39}	5272^{541}_{40}	5871^{694}_{52}
relative$_{lbbr}$	2450^{187}_{50}	10114^{4170}_{59}	3609^{86}_{38}	3711^{364}_{57}	12077^{455}_{50}	6467^{327}_{53}	709^{31}_{38}	2781^{82}_{35}	4862^{541}_{40}	5198^{694}_{47}
dynamic	1599^{109}_{24}	7646^{2450}_{16}	1857^{45}_{15}	2012^{203}_{22}	6297^{274}_{26}	3058^{185}_{17}	428^{16}_{24}	991^{50}_{19}	3041^{320}_{17}	2992^{406}_{21}
dynamic$_{bl}$	2999^{187}_{51}	12898^{4170}_{65}	3997^{86}_{36}	4506^{364}_{71}	9508^{455}_{51}	6095^{327}_{54}	717^{31}_{32}	1683^{82}_{33}	4631^{541}_{35}	5226^{694}_{47}
dynamic$_{br}$	2626^{187}_{44}	10006^{4170}_{36}	2526^{86}_{29}	3360^{364}_{51}	8966^{455}_{45}	4603^{327}_{46}	509^{31}_{38}	1326^{82}_{23}	4186^{541}_{37}	4234^{694}_{38}
dynamic$_{lb}$	1675^{109}_{24}	7747^{2450}_{16}	2643^{45}_{15}	2374^{203}_{22}	8679^{274}_{26}	3794^{185}_{17}	636^{16}_{24}	1777^{50}_{19}	3793^{320}_{17}	3680^{406}_{21}
dynamic$_{lbbl}$	3116^{187}_{51}	12918^{4170}_{65}	5511^{86}_{36}	4882^{364}_{71}	11961^{455}_{51}	6624^{327}_{54}	1013^{31}_{32}	2755^{82}_{33}	5518^{541}_{35}	6033^{694}_{47}
dynamic$_{lbbr}$	2536^{187}_{44}	10202^{4170}_{36}	4013^{86}_{29}	3638^{364}_{51}	11700^{455}_{45}	6309^{327}_{46}	762^{31}_{38}	2435^{82}_{23}	4836^{541}_{37}	5159^{694}_{38}
r-desc$_{bl}$	2008^{187}_{30}	8270^{4170}_{11}	1004^{86}_{7}	2917^{364}_{25}	4629^{455}_{28}	2818^{327}_{9}	363^{31}_{10}	464^{82}_{15}	3697^{541}_{19}	2908^{694}_{17}
r-desc$_{br}$	2186^{187}_{39}	8625^{4170}_{51}	1166^{86}_{23}	2905^{364}_{46}	4789^{455}_{39}	3322^{327}_{34}	321^{31}_{22}	515^{82}_{22}	3642^{541}_{33}	3052^{694}_{34}
r-desc$_{lbbl}$	1708^{187}_{30}	8177^{4170}_{11}	1314^{86}_{7}	2822^{364}_{25}	7054^{455}_{28}	3744^{327}_{9}	521^{31}_{10}	840^{82}_{15}	4249^{541}_{19}	3381^{694}_{17}
r-desc$_{lbbr}$	1839^{187}_{39}	8582^{4170}_{51}	1492^{86}_{23}	2749^{364}_{46}	7058^{455}_{39}	4548^{327}_{34}	474^{31}_{22}	935^{82}_{33}	4190^{541}_{33}	3541^{694}_{34}
l-desc$_{bl}$	1420^{187}_{24}	9906^{4170}_{56}	2100^{86}_{24}	2486^{364}_{30}	3985^{455}_{24}	4644^{327}_{32}	490^{31}_{24}	626^{82}_{14}	3482^{541}_{21}	3238^{694}_{27}
l-desc$_{br}$	915^{187}_{7}	6903^{4170}_{8}	1095^{86}_{10}	1313^{364}_{5}	3107^{455}_{7}	2666^{327}_{13}	320^{31}_{16}	430^{82}_{7}	2829^{541}_{6}	2175^{694}_{9}
l-desc$_{lbbl}$	1667^{187}_{24}	9496^{4170}_{56}	2605^{86}_{24}	3182^{364}_{30}	5862^{455}_{24}	4827^{327}_{32}	581^{31}_{24}	1036^{82}_{14}	4111^{541}_{21}	3707^{694}_{27}
l-desc$_{lbbr}$	1069^{187}_{7}	6748^{4170}_{8}	1610^{86}_{10}	1775^{364}_{5}	4689^{455}_{7}	3337^{327}_{13}	409^{31}_{16}	722^{82}_{7}	3480^{541}_{6}	2648^{694}_{9}
4tag$_{blin}$	1022^{187}_{4}	7690^{4170}_{4}	1660^{86}_{4}	1590^{364}_{4}	5397^{455}_{4}	3134^{327}_{4}	217^{31}_{4}	853^{82}_{4}	3239^{541}_{4}	2755^{694}_{4}
4tag$_{brin}$	1037^{187}_{4}	7157^{4170}_{4}	1389^{86}_{4}	1435^{364}_{4}	5201^{455}_{4}	2855^{327}_{4}	172^{31}_{4}	791^{82}_{4}	3220^{541}_{4}	2584^{694}_{4}
4tag$_{blpr}$	4563^{3801}_{44}	16884^{15465}_{72}	8360^{5652}_{42}	3685^{3239}_{52}	5627^{2950}_{45}	10982^{7644}_{58}	6557^{4191}_{40}	1497^{795}_{27}	4066^{2853}_{40}	6913^{5177}_{47}
4tag$_{brpr}$	2329^{1656}_{15}	14629^{12960}_{17}	5655^{3203}_{20}	3054^{2511}_{11}	5635^{2209}_{15}	6554^{3557}_{24}	3849^{2101}_{30}	1036^{475}_{13}	4075^{2440}_{13}	5202^{3457}_{18}
4tag$_{blpo}$	25078^{22938}_{141}	23525^{18957}_{52}	3524^{1156}_{21}	23607^{20093}_{98}	15699^{10036}_{72}	6149^{2370}_{34}	739^{309}_{37}	3457^{2050}_{38}	8082^{5980}_{76}	12207^{9321}_{63}
4tag$_{brpo}$	40731^{38920}_{341}	28035^{25297}_{433}	4342^{2602}_{54}	27877^{26071}_{548}	14112^{10560}_{300}	8434^{5746}_{329}	1328^{637}_{49}	4834^{3617}_{80}	7938^{5916}_{235}	15292^{13374}_{263}
att-jxt	2569^{409}_{29}	5794^{4152}_{11}	1554^{337}_{6}	3893^{974}_{32}	12150^{2286}_{27}	2611^{949}_{9}	287^{90}_{9}	1795^{254}_{14}	3464^{1253}_{19}	3791^{1189}_{17}
att-jxt$_{bl}$	6474^{1222}_{29}	17219^{12712}_{11}	3376^{1036}_{6}	13024^{3772}_{32}	17133^{3903}_{27}	5342^{2255}_{9}	521^{179}_{9}	3021^{527}_{14}	8015^{3652}_{19}	8236^{3251}_{17}
att-jxt$_{br}$	5942^{762}_{55}	27518^{7999}_{60}	5643^{726}_{24}	15569^{2230}_{66}	23246^{3811}_{52}	12488^{2017}_{49}	566^{146}_{26}	3969^{446}_{21}	9277^{2446}_{58}	11580^{2287}_{46}

Note that while larger or smaller label sets may, in principle, affect the learnability of an encoding and offer potential advantages, there is no necessary correlation between label set size and performance.

To provide a more nuanced view beyond raw label counts, as well as easier visualization, Figure 12 presents log-log plots of the rank-frequency distributions for the different linearization strategies on the Penn Treebank (PTB) dataset. Figures 12(a), (b), and (c) show the distribution of the encoding-specific field s_i for encodings of the common ancestor, directional depth-based, and transition-based families, respectively; while Figure 12(d) depicts the distribution of labels as a whole for all encodings. The plotted distributions correspond to the unbinarized, look-ahead variant of each encoding, except in the case of the tetra-tagging and directional encodings (which require binarization), where right binarization is used.

Table 8
General statistics for the selected treebanks, showing key structural characteristics as an average and standard deviation subscript. The *branching factor* indicates the average number of children per node. The *collapsed branching factor* column indicates the average number of children per node for the trees obtained after collapsing unary chains. The average sentence length represents the average number of words per sentence. The average tree depth depicts the average number of levels across all the trees in the treebanks. Finally, the percentages of right and left branches in the tree are calculated by performing an in-order traversal. During this traversal, each branch is checked to see if it extends to the right or left of its parent node. Standard deviations are given as subscripts.

Dataset	Lang	Branching factor	Collapsed Br. factor	Avg. sentence length	Avg. tree depth	Branch direction		Unique Nonterminals
						Right (%)	Left (%)	
PTB	en	$2.09_{0.39}$	$2.44_{0.28}$	$23.64_{11.09}$	$8.85_{3.93}$	$55.39_{5.19}$	$44.61_{5.19}$	26
SPMRL	eu	$1.79_{0.17}$	$2.38_{0.14}$	$13.15_{5.92}$	$5.75_{1.34}$	$41.81_{4.40}$	$58.19_{4.40}$	24
	fr	$2.42_{0.28}$	$2.52_{0.25}$	$30.35_{17.53}$	$8.55_{3.09}$	$56.32_{6.40}$	$43.68_{6.40}$	97
	de	$1.88_{0.60}$	$2.71_{0.68}$	$17.48_{11.14}$	$3.41_{1.24}$	$69.52_{10.29}$	$30.48_{10.29}$	315
	he	$1.47_{0.04}$	$2.25_{0.04}$	$23.98_{13.71}$	$10.27_{3.34}$	$31.21_{2.28}$	$68.79_{2.28}$	242
	hu	$2.01_{0.15}$	$2.78_{0.16}$	$23.05_{11.99}$	$6.13_{1.49}$	$48.42_{4.16}$	$51.58_{4.16}$	210
	ko	$1.73_{0.14}$	$2.12_{0.12}$	$12.51_{5.12}$	$8.75_{2.08}$	$40.10_{3.90}$	$59.90_{3.90}$	12
	pl	$1.29_{0.07}$	$2.51_{0.04}$	$10.17_{5.21}$	$7.98_{2.36}$	$24.75_{2.58}$	$75.25_{2.58}$	35
	sv	$2.12_{0.23}$	$2.40_{0.19}$	$16.73_{10.43}$	$7.02_{2.64}$	$49.51_{8.05}$	$50.49_{8.05}$	200

In general terms, the distribution for most encodings resembles a Zipf-Mandelbrot distribution, and reminds of distributions of syntactic categories observed in previous work (cf. Figure 6 of Piantadosi [2014]); but there are considerable differences between encodings regarding the degree of head flatness and tail steepness.

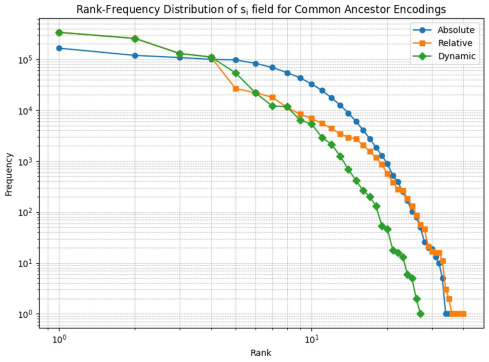
Figure 12(a) reveals that, among common-ancestor linearizations, the dynamic encoding produces the most compact distribution. The relative encoding has larger frequencies in the top ranks than the absolute one, which can make these frequent values more learnable, but also has a longer tail.

Figure 12(b) shows a more distinct pattern for the descendant encodings. The left-descendant curve shows a much steeper drop-off and a shorter tail compared to the right-descendant encoding. This indicates that the right-descendant encoding produces a substantially larger set of rare, low-frequency labels. This observation aligns with the structural properties of the Penn Treebank, which is predominantly right-branching. The prevalence of right-branching structures leads to longer chains of right children, which results in a wider range of labels for the right-descendant encoding.

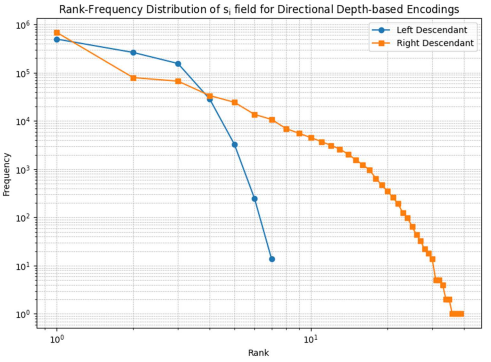
The transition-based encodings, shown in Figure 12(c), also show dramatic differences. In-order tetra-tagging is naturally the most compact, as could be expected, having only four possible labels. Tetra-tag pre-order and post-order variants exhibit longer and flatter tails, confirming our earlier finding that the lack of paired alignment in these variants results in sparsity. For its part, the attach-juxtapose encoding also shows a rather sparse distribution.

Figure 12(d) (which includes the full label, and not only the encoding-specific component) shows the big picture, where left-descendant and in-order tetra-tagging display the most compact distribution and the post-order tetra-tagging and attach-juxtapose have the sparsest ones on the PTB dataset.

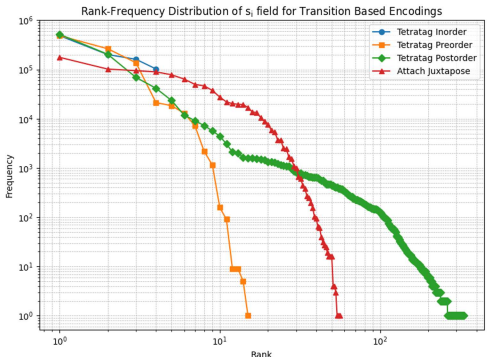
We next evaluate the performance of the studied encodings.



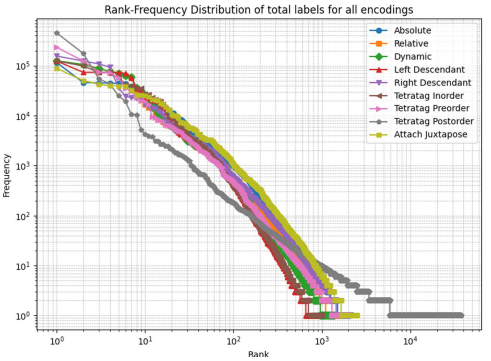
(a) Distribution of the s_i component for common ancestor encodings.



(b) Distribution of the s_i component for directional depth-based encodings.



(c) Distribution of the s_i component for transition-based encodings.



(d) Distribution of the whole label space for all encodings.

Figure 12
Rank-frequency distribution across different linearization strategies for the English Penn Treebank dataset. Subfigures (a), (b), and (c) plot the distribution of the label-specific s_i field only for the three different encoding families; and subfigure (d) plots the distribution of the complete label across all families.

6.2 Accuracy Results

We now present the parsing performance results obtained when using the different encodings. We will first present results with the depth-based methods, followed by the transition-based encodings.

6.2.1 Results for the Depth-based Encodings. Table 9 summarizes the results for the PTB and SPMRL treebanks, comparing the original `absolute`, `relative`, and `dynamic` encodings. We also include the proposed variants: left and right binarizations (bl and br) and the look-behind version (lb).

The results show that the choice of how we represent the number of common ancestors (i.e., choice between `absolute`, `relative`, or `dynamic`) has a rather mild influence in performance; with the `dynamic` encoding typically outperforming the others by around 0.1 to 0.6 points of F1-score on average, all other things being equal. This strongly

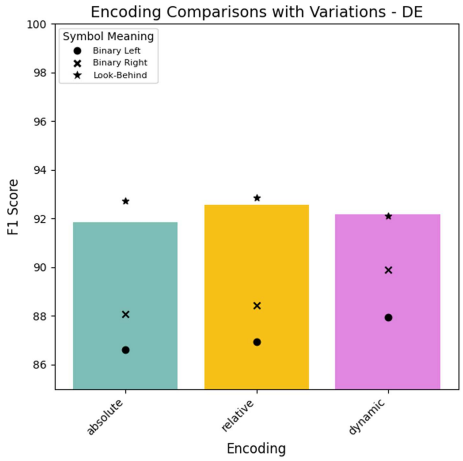
Table 9
Overview of F1-scores for common-ancestor depth-based encoding methods in constituent parsing, evaluated on the English Penn Treebank and SPMRL datasets. The subscript acronyms represent variants of the original depth-based encodings, as described in §4.1.3: binary right (br), binary left (bl), and look-behind (lb).

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
absolute	95.12	91.85	95.03	87.08	92.56	86.80	88.12	96.59	90.80	91.55
$\text{absolute}_{\text{br}}$	94.01	88.06	94.26	85.06	92.12	85.68	87.62	96.22	90.55	90.40
$\text{absolute}_{\text{bl}}$	94.15	86.61	94.30	**88.61**	93.16	85.15	89.21	96.33	90.32	90.87
$\text{absolute}_{\text{lb}}$	95.15	92.72	**95.13**	86.68	92.03	86.12	**92.66**	96.70	**91.61**	92.09
relative	94.99	92.56	93.27	87.10	92.72	88.15	87.21	96.91	89.15	91.34
$\text{relative}_{\text{br}}$	94.12	88.42	93.82	84.92	92.89	86.61	86.61	96.12	89.04	90.28
$\text{relative}_{\text{bl}}$	92.11	86.94	90.13	86.01	93.10	87.70	89.95	96.95	88.12	90.11
$\text{relative}_{\text{lb}}$	95.28	**92.84**	93.93	87.02	92.70	88.98	91.03	97.18	89.33	92.03
dynamic	**95.54**	92.16	94.78	87.36	**93.88**	**90.03**	87.60	**97.35**	88.85	91.95
$\text{dynamic}_{\text{br}}$	94.92	89.89	94.02	85.12	92.65	89.65	85.13	97.21	88.65	90.80
$\text{dynamic}_{\text{bl}}$	93.02	87.94	92.33	87.13	93.32	89.35	89.35	97.30	88.33	90.90
$\text{dynamic}_{\text{lb}}$	95.14	92.12	94.61	87.21	93.82	89.05	91.28	97.31	89.26	**92.20**

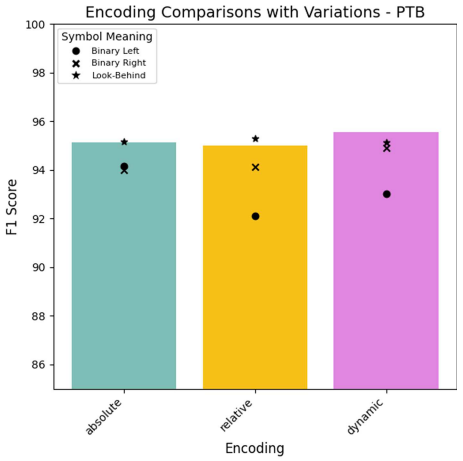
contrasts with previous work on constituent parsing as sequence labeling, which used BiLSTMs as encoders (Gómez-Rodríguez and Vilares 2018; Vilares, Abdou, and Søgaard 2019). Under this setup, the absolute encoding was not considered competitive due to sparsity, to the point that it was not even included in final experiments due to weak preliminary results. Our results show that the more recent Transformer-based encoders have made it viable to learn the absolute representation. In fact, it even outperforms the relative encoding on average, and obtains the overall best results across the three encodings on several treebanks. In this respect, a similar finding was reported by Vacareanu et al. (2020) in the context of dependency parsing. They revisited previously proposed encoding strategies, this time using BERT instead of BiLSTMs, and observed that some simple encodings that had failed with BiLSTMs were effective when combined with BERT.

Another relevant observation is that the choice of encoding variant, which had been overlooked in previous literature, has a greater influence than the choice of encoding itself, with the difference between the best and worst variant of each encoding being in the range of 1 to 2 F1-score points. In this respect, binarization (regardless of direction) seems to be harmful on average, although it can be worth considering on a treebank-specific basis—for example, for the Korean treebank, left binarization is beneficial, likely because it is the treebank with the smallest number of unique nonterminals (12; see Table 8) so binarizing it does not cause as much of an increase in the number of nonterminals (and hence label set size) as in other languages (as can be checked in Table 7). On the other hand, the look-behind variant is beneficial on average, obtaining the best average F1-scores regardless of encoding, although the effect is treebank-specific with large performance boosts on Korean (about 4 points, consistent across all encodings) and only slight differences in the other datasets.

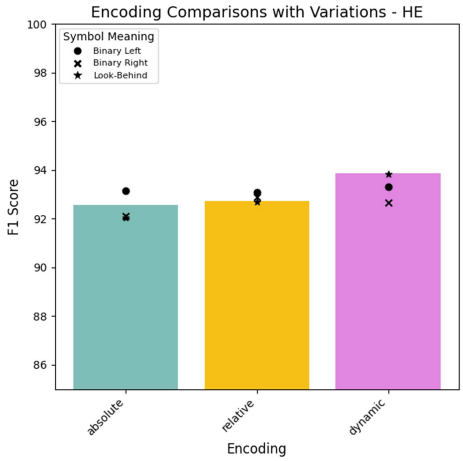
To analyze some of the results more visually, Figure 13 presents F1-score plots for selected datasets that exhibit specific structural characteristics we hypothesized



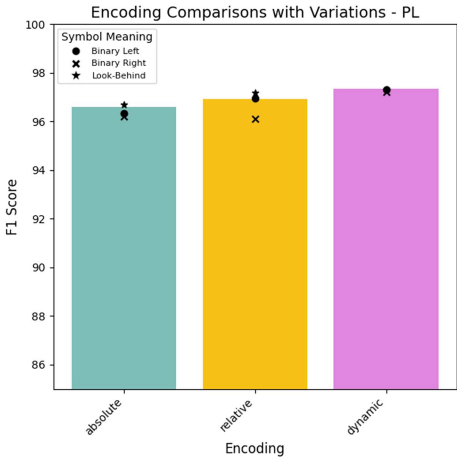
(a) Plot for the German SPMRL dataset



(b) Plot for the English PTB dataset



(c) Plot for the Hebrew SPMRL dataset



(d) Plot for the Polish SPMRL dataset

Figure 13
F1-scores obtained using three depth-based encoding strategies (absolute, relative, dynamic) across four treebanks: German SPMRL (the most right-branching), Polish SPMRL (the most left-branching), Hebrew SPMRL (with the highest average tree depth), and English PTB. Each bar shows performance under three variation settings: binary left, binary right, and look-behind.

could impact the encodings: the most right-branching (de_{SPMRL}), the most left-branching (pl_{SPMRL}), the deepest trees (he_{SPMRL}), and the shallowest trees (de_{SPMRL}). As an additional point of reference, we also include the F1-score plot for the English dataset (PTB). One insight we can derive from the plots is that the binarized depth-based approaches perform worse on the right-branching treebanks. This is particularly noticeable for German as the most right-branching treebank but can also be seen in English, which is also right-branching, when compared with the other two (left-branching) datasets in the plot. If we instead look at the effect of the look-behind encoding, it seems more related to depth: it is most effective in the German (shallowest) treebank, and least effective in the Hebrew (deepest) one, with the other two sitting in between. Finally, in terms of comparison between encodings, the benefit of `dynamic` over `absolute` and

Table 10
Overview of F1-scores for directional depth-based encoding methods in constituent parsing, evaluated on the English Penn Treebank and SPMRL datasets. Notation as in Table 9.

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
`l-desc`$_{\text{br}}$	**95.14**	**93.11**	**94.50**	**87.86**	**93.74**	88.62	89.66	**97.72**	**90.62**	**92.33**
`l-desc`$_{\text{br,lb}}$	93.85	91.97	91.78	82.79	90.45	87.12	**90.23**	96.41	89.45	90.45
`l-desc`$_{\text{bl}}$	87.21	86.02	88.23	80.97	87.34	82.15	84.45	92.61	86.10	86.12
`l-desc`$_{\text{bl,lb}}$	81.65	80.78	81.91	77.75	80.30	78.92	79.50	85.91	81.23	80.88
`r-desc`$_{\text{br}}$	93.35	92.42	92.48	86.95	93.12	**88.89**	88.21	95.65	90.45	91.28
`r-desc`$_{\text{br,lb}}$	93.12	90.45	91.23	82.85	90.32	87.00	89.78	96.15	89.12	90.00
`r-desc`$_{\text{bl}}$	85.32	84.10	86.45	78.92	85.67	80.78	82.15	90.23	83.45	84.12
`r-desc`$_{\text{bl,lb}}$	80.12	79.35	80.89	76.21	79.45	77.92	78.55	84.12	79.87	79.61

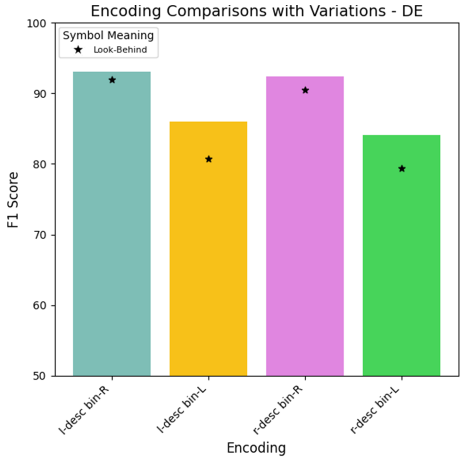
`relative` also seems to increase with treebank depth, being largest in Hebrew. This confirms that the `dynamic` encoding is especially helpful in cases where large relative depth differences appear, as we had hypothesized.

Table 10 shows the results for the new left-descendant (`l-desc`) and right-descendant (`r-desc`) encodings, in their left-binarized (bl) and right-binarized (br) versions, and again including also the look-behind (lb) variant. One of the most notable results is that right binarization consistently and greatly outperforms left binarization across all encodings and datasets. For example, in PTB, `r-desc`$_{\text{br}}$ obtains 93.35 F1-score, while its left-binarized version, `r-desc`$_{\text{bl}}$, drops 8 points down to 85.32. Similar performance gaps appear in the SPMRL treebanks, with left binarization reducing F1-scores by 4 to 10 points. This effect did not appear in common-ancestor depth-based encodings, and cannot be explained by label set size (recall from Table 2 that right binarization yielded more compact label spaces with the `l-desc` encoding, but the opposite with `r-desc`). The reason for the higher sensitivity of directional encodings to binarization direction, compared to common-ancestor encodings, might be because directional depth-based encodings are directly based on the directionality of the path from a word to its ancestors, counting left or right child links, making them highly sensitive to the local geometry of the tree and the paths created by binarization. In contrast, common-ancestor encodings simply count the number of shared ancestors between two adjacent words, which is a more direction-agnostic measure: It does not matter if the path to those ancestors involves left or right turns.

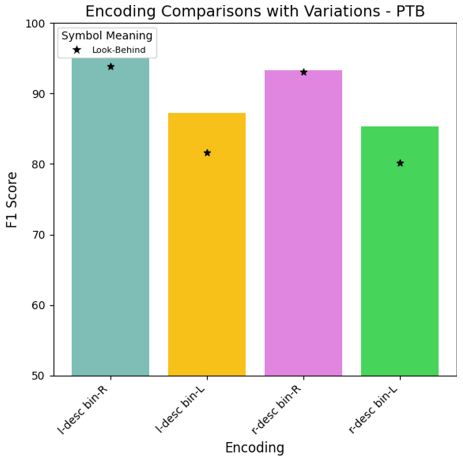
Regarding how look-behind parsing affects directional encodings, in general the effect is negative with substantial accuracy drops (in the 1 to 6 F1-score range on average). Again, Korean is the most favorable language for this variant, achieving improvements over the standard look-ahead version when the treebank is right-binarized; but in this case the consistent drop in all other languages penalizes the cross-lingual average.

Finally, it is also worth remarking that, whereas the performance of directional depth-based encodings is more brittle to variant choice than that of common ancestor encodings; the `l-desc`$_{\text{br}}$ outperforms all of those in terms of average F1—and in fact, obtains the best average F1 across all encodings and variants studied in this article.

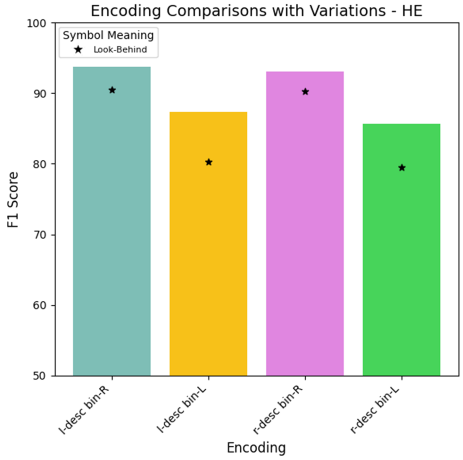
Again, we present a graphical analysis of the treebanks with extreme depth and branching values (plus English as a reference) in Figure 14. Apart from confirming the general trends noted in the table, we can see that left binarization is more harmful in



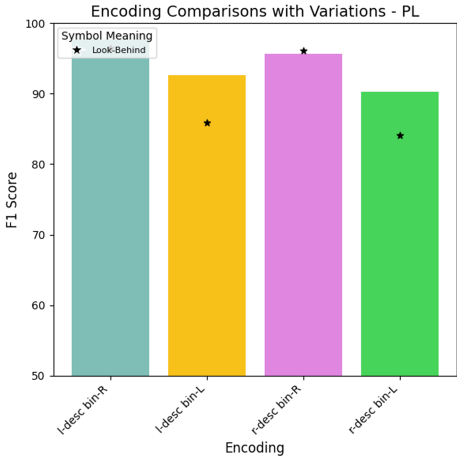
(a) Plot for the German SPMRL dataset



(b) Plot for the English PTB dataset



(c) Plot for the Hebrew SPMRL dataset



(d) Plot for the Polish SPMRL dataset

Figure 14
F1-score obtained using left and right-descendant (l-desc and r-desc) encodings with both binarization variants (bin-R and bin-L) as well as the look-behind variant for the English PTB treebank, the most right branching treebank (German SPMRL), the most left leaning treebank (Polish SPMRL), and the dataset with the highest depth (Hebrew SPMRL).

right-branching treebanks. However, we do not observe any clear trend regarding how treebank characteristics affect the influence (in this case, always negative) of using the look-behind variant.

6.2.2 Results for the Transition-based Encodings. Table 11 presents the F1-scores for transition-based encodings and their variants. A first observation is that the pre- and post-order traversal variants of tetra-tagging (`4tag`) lead to uncompetitive scores, suggesting that this traversal order is less effective for transition-based encodings. This is consistent with our observation of high label sparsity with these traversals, as seen in Table 3, which is explained by the lack of paired alignment as labels in these variants do not encode a constant number of actions or nonterminals (recall also Figure 8). It

Table 11
Overview of F1-scores for transition-based encoding methods in constituent parsing, evaluated on the English Penn Treebank and SPMRL datasets. Notation as in Table 9.

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
$\texttt{4tag}_{\texttt{brin}}$	94.68	**93.16**	94.53	85.34	89.16	**90.12**	**91.01**	**97.43**	91.15	91.84
$\texttt{4tag}_{\texttt{blin}}$	90.31	87.24	89.15	76.13	78.98	88.35	82.15	86.23	78.23	84.09
$\texttt{4tag}_{\texttt{brpr}}$	91.85	91.20	92.30	**93.20**	81.90	84.50	85.10	89.50	**93.85**	89.27
$\texttt{4tag}_{\texttt{blpr}}$	87.10	87.05	86.20	88.00	73.00	74.80	83.50	80.50	82.90	82.56
$\texttt{4tag}_{\texttt{brpo}}$	84.50	83.50	61.50	82.00	58.20	70.50	90.00	81.20	63.80	75.02
$\texttt{4tag}_{\texttt{blpo}}$	81.00	80.10	57.50	77.00	52.00	62.90	88.00	73.80	57.50	69.98
`att-jxt`	**95.76**	92.42	94.62	87.91	**92.06**	89.99	90.04	97.23	89.14	**92.13**
$\texttt{att-jxt}_{\texttt{br}}$	94.90	90.85	93.97	84.85	91.14	86.58	89.00	96.41	87.74	90.60
$\texttt{att-jxt}_{\texttt{bl}}$	94.38	88.62	**94.70**	86.63	91.19	86.91	88.06	96.61	88.26	90.60

is also consistent with the findings of Amini and Cotterell (2022) when comparing the in-order, pre-order, and post-order variants of tetra-tagging. While we also performed experiments with pre-order and post-order tetra-tagging enforcing paired alignment instead of zero deviation, the results (not shown in the table) yielded even worse accuracy. On the other hand, if we focus on the in-order traversal, we can see that the performance is consistently and considerably better if the treebank is right-binarized with respect to left binarization, similarly to what happened with directional depth-based encodings (Table 10). In this respect, it is worth remarking that while tetra-tagging and directional depth-based encodings belong to totally different encoding families, they do have some similarities (namely, they are based on representing the directionality of children), which relate to this pattern.

In the case of attach-juxtapose (`att-jxt`), the best performance by far is obtained with the unbinarized treebank. Binarization has a considerable negative impact, consistent with the large increases in label set size previously observed in Table 4 caused by the fact that the potential number of labels that can be generated with this encoding is quadratic, not linear, with respect to the number of nonterminals. If we compare both binarization directions, the predominant branching direction seems to play a role in encoding effectiveness. Languages with a strong right-branching tendency, such as German and English, show a preference for right binarization, whereas highly left-branching languages, such as Polish, Hebrew, and Basque work better with left binarization. However, for every language except Basque, lack of binarization is clearly better than even the best binarization.

Because in this case it is clear what the variant of choice should be for each encoding ($\texttt{4tag}_{\texttt{brin}}$ for tetra-tagging, and the unbinarized `att-jxt` for attach-juxtapose), we now compare them against each other. On average, both encodings show similar results, although attach-juxtapose has a slight advantage of 0.25 points in average F1-score.

Treebanks with higher average depth, such as Hebrew (10.27), English (8.85), or French (8.55), tend to favor attach-juxtapose. This is likely due to its ability to model deeper structures more effectively, as the algorithm is designed in such a way that new branches can be attached at any arbitrary height of a long tree spine. The Korean dataset is an exception to this pattern (with tetra-tagging achieving higher accuracy while having a similar tree depth as the ones mentioned above). This could be due to

label set size, as the Korean treebank generates a particularly small set of 172 labels with in-order tetra-tagging (see Table 3), while the label set with attach-juxtapose is considerably larger (Table 4).

The performance of the `att-jxt` encoding also highlights that, while label set size can be a factor in performance, particularly when we consider closely-related encodings that are directly comparable (hence the negative impact of binarization in this encoding, as mentioned above), it is far from providing the whole picture. On the English Penn Treebank, the `att-jxt` encoding produces the best overall accuracy across all encodings tested, despite exhibiting a rather sparse label distribution on this dataset (cf. Figure 12(d)). In this case, it appears that the less favorable distribution of the label set is counterbalanced by other factors, such as the suitability of the `att-jxt` logic for the relatively deep trees of the PTB (since it allows attaching new branches at any arbitrary height along a long rightmost spine) and the right-branching predominance of English syntax (the attach and juxtapose operations naturally make the tree grow rightward).

7. Discussion

We have introduced a wide range of encodings for constituent parsing as sequence labeling, including existing encodings, novel encodings that had not been implemented before (directional depth-based encodings), and adaptations of parsers from other paradigms that had not previously been cast under the sequence-labeling framework (attach-juxtapose). In addition, for the first time we have considered different variants of each encoding, according to whether and how the treebank is binarized and the directional context used to generate each label, distinguishing between using the next word or the previous word as a reference.

From a theoretical point of view, we have uncovered some properties of encodings and relations between them. From a practical standpoint, we have compared all the encodings and variants on a wide range of constituent treebanks, both in terms of label set size and of parsing performance. For the latter, we have implemented the parsers under a uniform setup with a modern Transformer-based encoder (XML-RoBERTa), which allows both to make a fair and homogeneous comparison and to determine how the practical viability of different encodings has evolved with the machine learning advances of the last few years.

The results of our experiments show the following new practical insights:

- The improvements in neural encoders in the last few years have widened the range of encodings that can be used to obtain accurate constituent parsers. In particular, the absolute encoding, which was not learnable with BiLSTM-based encoders (Gómez-Rodríguez and Vilares 2018), obtains competitive accuracy when implemented with XLM-RoBERTa, even leading for some treebanks (our overall best results on Basque and Korean are obtained with the absolute encoding).
- The choice of variant (left, right, or no binarization; and look-ahead or look-behind), which had not even been considered in previous literature, generally has an even greater impact on accuracy than the choice of encoding. Given the influence of binarization direction on performance, exploring methods to determine an optimal mix of left and right binarizations for each language presents an interesting direction for future work.

- If we choose the right variant for each encoding, the three encoding families that we have considered (common ancestor, directional depth-based, and transition-based) are roughly tied in terms of average F1-score across our tested treebanks, showing that all families can produce competitive results.
- Label set size is an important factor affecting encoding performance (with variants with especially large label spaces, like pre-order and post-order tetra-tagging, clearly lagging behind); but is far from explaining it on its own. We have seen how the best encoding for a given treebank can be influenced by other factors such as branching direction, binarization direction, number of nonterminals, and tree depth.
- Overall, the most robust encoding in terms of average F1-score across our diverse set of treebanks is the left-descendant encoding, which had not been implemented before. This encoding also produces the most compact label set size on average.

To conclude, it is worth highlighting that parsing as sequence labeling can provide results that are rather close to the state of the art. For example, on the Penn Treebank, we obtain 95.76 F1-score with the `att-jxt` encoding. The current state-of-the-art model obtains 96.48 (Yang and Tu 2023), a difference of 0.72. However, that result and others that obtain similar accuracies (e.g., Tian et al. 2020; Wang and Utiyama 2024), were obtained with an English-specific encoder (XLnet) and hyperparameter tuning. In contrast, our results were obtained within a uniform experimental setup using the multilingual XLM-RoBERTa encoder across all languages, without any hyperparameter optimization. We relied solely on an off-the-shelf sequence labeling library (MaChAmp) with its default settings, thus reflecting the practical scenario of NLP practitioners who use parsing primarily as a tool rather than a primary research focus.

In comparison to any of those state-of-the-art models, the sequence labeling approach provides efficiency (assigning a linear number of labels, while said models score at least a quadratic number of spans, with some having cubic components), flexibility, and ease of use (using an off-the-shelf labeler, rather than ad hoc algorithms); thus proving a solid practical alternative when constituent parsing is needed.

Limitations

All experiments were conducted using a single neural sequence labeling architecture (based on the XLM-RoBERTa encoder) without any task-specific hyperparameter tuning. Although this choice was deliberate to ensure homogeneity and reflect realistic usage scenarios for off-the-shelf tools, it is also a limitation, as our findings may not fully generalize to other architectures or settings, and do not reflect the maximum absolute performance that could be obtained if careful hyperparameter tuning were applied for each encoding. Future work could explore how different encoders, training setups, or hyperparameter tuning affect the results, but this is outside our current resources.

Regarding the treebanks used for evaluation, although our selection spans nine languages and is as linguistically diverse as we could find, it has a clear Indo-European bias (5 Indo-European and 4 non-Indo-European languages). This is due to the limited availability of annotated constituency treebanks when compared to, e.g., dependency treebanks.

Finally, in this work we focused on using an off-the-shelf, easy-to-integrate sequence labeling architecture (MaChAmp), rather than an implementation optimized for speed. Table 13 in the Appendix reports the tokens-per-second performance for a selection of encodings, which still remains satisfactory.

Ethics Statement

We do not observe ethical implications in our work. Our research focuses on technical aspects of constituent parsing as sequence labeling, which are primarily computational and linguistic challenges.

Appendix

Table 12 presents the accuracy results for all tested encodings and variants, and Table 13 reports the parsing speed (words per second) for the main representative of each encoding, using CPU only in the MaChAmp sequence labeler.

Table 12
Overview of F1-scores for all the developed encoding methods in constituent parsing, evaluated on the English Penn Treebank and SPMRL datasets. Best results per encoding family (common-ancestor, directional, transition-based) are shown in bold; overall best results for each treebank are underlined.

Encoding	PTB	SPMRL								Average
		de	eu	fr	he	hu	ko	pl	sv	
Depth-based Encoding (Common-Ancestor Family)										
absolute	95.12	91.85	95.03	87.08	92.56	86.80	88.12	96.59	90.80	91.55
absolute$_{br}$	94.01	88.06	94.26	85.06	92.12	85.68	87.62	96.22	90.56	90.40
absolute$_{bl}$	94.15	86.61	94.30	**88.61**	93.16	85.15	89.21	96.33	90.32	90.87
absolute$_{lb}$	95.15	92.72	**95.13**	86.68	92.03	86.12	**92.66**	96.70	**91.61**	92.09
relative	94.99	92.56	93.27	87.10	92.72	88.15	87.21	96.91	89.15	91.34
relative$_{br}$	94.12	88.42	93.82	84.92	92.89	86.61	86.61	96.12	89.04	90.28
relative$_{bl}$	92.11	86.94	90.13	86.01	93.10	87.70	89.95	96.95	88.12	90.11
relative$_{lb}$	95.28	**92.84**	93.93	87.02	92.70	88.98	91.03	97.18	89.33	92.03
dynamic	**95.54**	92.16	94.78	87.36	**93.88**	**90.03**	87.60	**97.35**	88.85	91.95
dynamic$_{br}$	94.92	89.89	94.02	85.12	92.65	89.65	85.13	97.21	88.65	90.80
dynamic$_{bl}$	93.02	87.94	92.33	87.13	93.32	89.35	89.35	97.30	88.33	90.90
dynamic$_{lb}$	95.14	92.12	94.61	87.21	93.82	89.05	91.28	97.31	89.26	**92.20**
Directional Depth-based Encoding Family										
l-desc$_{br}$	**95.14**	**93.11**	**94.50**	**87.86**	**93.74**	88.62	89.66	**97.72**	**90.62**	**92.33**
l-desc$_{br,lb}$	93.85	91.97	91.78	82.79	90.45	87.12	**90.23**	96.41	89.45	90.45
l-desc$_{bl}$	87.21	86.02	88.23	80.97	87.34	82.15	84.45	92.61	86.10	86.12
l-desc$_{bl,lb}$	81.65	80.78	81.91	77.75	80.30	78.92	79.50	85.91	81.23	80.88
r-desc$_{br}$	93.35	92.42	92.48	86.95	93.12	**88.89**	88.21	95.65	90.45	91.28
r-desc$_{br,lb}$	93.12	90.45	91.23	82.85	90.32	87.00	89.78	96.15	89.12	90.00
r-desc$_{bl}$	85.32	84.10	86.45	78.92	85.67	80.78	82.15	90.23	83.45	84.12
r-desc$_{bl,lb}$	80.12	79.35	80.89	76.21	79.45	77.92	78.55	84.12	79.87	79.61
Transition-based Encoding Family										
4tag$_{brin}$	94.68	**93.16**	94.53	85.34	89.16	**90.12**	**91.01**	**97.43**	91.15	91.84
4tag$_{blin}$	90.31	87.24	89.15	76.13	78.98	88.35	82.15	86.23	78.23	84.09
4tag$_{brpr}$	91.85	91.20	92.30	**93.20**	81.90	84.50	85.10	89.50	**93.85**	89.27
4tag$_{blpr}$	87.10	87.05	86.20	88.00	73.00	74.80	83.50	80.50	82.90	82.56
4tag$_{brpo}$	84.50	83.50	61.50	82.00	58.20	70.50	90.00	81.20	63.80	75.02
4tag$_{blpo}$	81.00	80.10	57.50	77.00	52.00	62.90	88.00	73.80	57.50	69.98
att-jxt	**95.76**	92.42	94.62	87.91	**92.06**	89.99	90.04	97.23	89.14	**92.13**
att-jxt$_{br}$	94.90	90.85	93.97	84.85	91.14	86.58	89.00	96.41	87.74	90.60
att-jxt$_{bl}$	94.38	88.62	**94.70**	86.63	91.19	86.91	88.06	96.61	88.26	90.60

Table 13
Full parsing time (sequence labeling plus decoding of the tree) for each encoding in words per second, using MaChAmp sequence labeler in an i9-13900K CPU.

Encoding	PTB	SPMRL								Average$_{std}$
		de	eu	fr	he	hu	ko	pl	sv	
absolute	2144.6	2773.0	2062.7	2339.2	1559.5	2240.1	2046.5	1353.1	2681.6	$2133.4_{436.4}$
relative	2201.0	2652.0	2157.3	2218.6	1412.5	1921.2	2013.2	1272.8	2609.9	$2050.9_{443.1}$
dynamic	2200.4	2655.7	2220.8	2198.1	1432.9	2042.0	1920.8	1408.6	2700.1	$2086.6_{429.5}$
l-desc$_{br}$	1857.9	2443.7	1943.3	1924.1	1131.8	2146.2	1470.6	1363.4	1460.4	$1749.1_{396.0}$
r-desc$_{br}$	1993.8	2560.3	1879.6	2258.1	1863.9	2251.8	1958.1	1382.8	2351.7	$2055.6_{325.8}$
4tag$_{brin}$	1956.7	2332.0	1658.6	1946.8	1356.9	2046.8	1765.7	1462.3	1801.1	$1814.1_{282.8}$
att-jxt	1438.0	1698.5	1432.4	1839.4	677.4	1029.4	575.9	648.3	2776.3	$1346.2_{671.4}$

Acknowledgments

We acknowledge grants GAP (PID2022-139308OA-I00) funded by MICIU/AEI/10.13039/501100011033/ and ERDF, EU; LATCHING (PID2023-147129OB-C21) funded by MICIU/AEI/10.13039/501100011033 and ERDF, EU; PREP2023-001786 funded by MICIU/AEI/10.13039/501100011033 and ESF+ (predoctoral training grant associated to project PID2023-147129OB-C21); and TSI-100925-2023-1 funded by Ministry for Digital Transformation and Civil Service and "NextGenerationEU" PRTR; as well as funding by Xunta de Galicia (ED431C 2024/02), 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). We are grateful to the funding of Consellería de Educación, Ciencia, Universidades e Formación Profesional (Xunta de Galicia - Convenio para o desenvolvemento de accións estratéxicas de I+D+i 2025-2026).

References

Abney, Steven P. 1992. Parsing by chunks. *Principle-based parsing: Computation and Psycholinguistics*, pages 257–278. https://doi.org/10.1007/978-94-011-3474-3_10

Alonso, Miguel A., Carlos Gómez-Rodríguez, and Jesús Vilares. 2021. On the use of parsing for named entity recognition. *Applied Sciences*, 11(3):1090. https://doi.org/10.3390/app11031090

Amini, Afra and Ryan Cotterell. 2022. On parsing as tagging. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8884–8900. https://doi.org/10.18653/v1/2022.emnlp-main.607

Amini, Afra, 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. https://doi.org/10.18653/v1/2023.acl-short.124

Anderson, Mark and Carlos Gómez-Rodríguez. 2021. A modest Pareto optimisation analysis of dependency parsers in 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 119–130. https://doi.org/10.18653/v1/2021.iwpt-1.12

Blank, Idan, Zuzanna Balewski, Kyle Mahowald, and Evelina Fedorenko. 2016. Syntactic processing is distributed across the language system. *NeuroImage*, 127:307–323. https://doi.org/10.1016/j.neuroimage.2015.11.069, PubMed: 26666896

Brants, Thorsten. 2000. TnT – a statistical part-of-speech tagger. In *Sixth Applied Natural Language Processing Conference*, pages 224–231. https://doi.org/10.3115/974147.974178

Clark, Stephen. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19–24.

Clark, Stephen and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288. https://doi.org/10.3115/1220355.1220396

Coavoux, Maximin and Shay B. Cohen. 2019. Discontinuous constituency parsing with a stack-free transition system and a dynamic oracle. 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 204–217. https://doi.org/10.18653/v1/N19-1018

Conneau, A. 2019. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*. https://doi.org/10.18653/v1/2020.acl-main.747

Damonte, Marco and Emilio Monti. 2021. One semantic parser to parse them all: Sequence to sequence multi-task learning on semantic parsing datasets. In *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 173–184. https://doi.org/10.18653/v1/2021.starsem-1.16

Devlin, Jacob, 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. https://doi.org/10.18653/v1/N19-1423

Ezquerro, Ana, Carlos Gómez-Rodríguez, and David Vilares. 2023. On the challenges of fully incremental neural dependency parsing. *arXiv preprint arXiv:2309.16254*. https://doi.org/10.18653/v1/2023.ijcnlp-short.7

Ezquerro, Ana, Carlos Gómez-Rodríguez, and David Vilares. 2024. From partial to strictly incremental constituent parsing, Graham, Yvette and Matthew Purver, editors. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 225–233. https://doi.org/10.18653/v1/2024.eacl-short.21

Ezquerro, Ana, Carlos Gómez-Rodríguez, and David Vilares. 2025. Better benchmarking LLMs for zero-shot dependency parsing. In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pages 121–135.

Ezquerro, Ana, David Vilares, Anssi Yli-Jyrä, and Carlos Gómez-Rodríguez. 2025. Hierarchical bracketing encodings for dependency parsing as tagging. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18436–18450. https://doi.org/10.18653/v1/2025.acl-long.903

Fedorenko, Evelina, Idan Asher Blank, Matthew Siegelman, and Zachary Mineroff. 2020. Lack of selectivity for syntax relative to word meanings throughout the language network. *Cognition*, 203:104348. https://doi.org/10.1016/j.cognition.2020.104348, PubMed: 32569894

Gómez-Rodríguez, Carlos, 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. https://doi.org/10.18653/v1/2023.emnlp-main.393

Gómez-Rodríguez, Carlos, Michalina Strzyz, and David Vilares. 2020. A unifying theory of transition-based and sequence labeling parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3776–3793. https://doi.org/10.18653/v1/2020.coling-main.336

Gómez-Rodríguez, Carlos 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. https://doi.org/10.18653/v1/D18-1162

Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Joshi, Aravind K. and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*. https://doi.org/10.3115/991886.991912

Kitaev, Nikita 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. https://doi.org/10.18653/v1/2020.acl-main.557

Kodner, Jordan, Salam Khalifa, and Sarah Payne. 2023. Exploring linguistic probes for morphological generalization. *arXiv preprint arXiv:2310.13686*. https://doi.org/10.18653/v1/2023.emnlp-main.552

Lacroix, Ophélie. 2019. Dependency parsing as sequence labeling with head-based encoding and multi-task learning. In *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*, pages 136–143. https://doi.org/10.18653/v1/W19-7716

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

Li, Junhui, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697. https://doi.org/10.18653/v1/P17-1064

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

Lin, Boda, Zijun Yao, Jiaxin Shi, Shulin Cao, Binghao Tang, Si Li, Yong Luo, Juanzi Li, and Lei Hou. 2022. Dependency parsing via sequence generation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 7339–7353. https://doi.org/10.18653/v1/2022.findings-emnlp.543

Lin, Boda, Xinyi Zhou, Binghao Tang, Xiaocheng Gong, and Si Li. 2023. ChatGPT is a potential zero-shot dependency parser. *arXiv preprint arXiv:2310.16654*.

Lin, Yongjie, Yi Chern Tan, and Robert Frank. 2019. Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253. https://doi.org/10.18653/v1/W19-4825

Liu, Jiangming and Yue Zhang. 2017. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424. https://doi.org/10.1162/tacl_a_00070

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. https://doi.org/10.21236/ADA273556

Margueritte, Gaëtan, Daisuke Bekki, and Koji Mineshima. 2023. Multi-purpose neural network for French categorial grammars. In *Proceedings of the 15th International Conference on Computational Semantics*, pages 78–82.

Màrquez, Lluís, Pere Comas, Jesús Giménez, and Neus Català. 2005. Semantic role labeling as sequential tagging. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 193–196. https://doi.org/10.3115/1706543.1706579

Muñoz-Ortiz, Alberto, David Vilares, and Carlos Gómez-Rodríguez. 2023. Assessment of pre-trained models across languages and grammars. *arXiv preprint arXiv:2309.11165*. https://doi.org/10.18653/v1/2023.ijcnlp-main.23

Ninomiya, Takashi, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163. https://doi.org/10.3115/1610075.1610100

Piantadosi, Steven T. 2014. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21(5):1112–1130. https://doi.org/10.3758/s13423-014-0585-6, PubMed: 24664880

Ramshaw, Lance and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Rotman, Guy and Roi Reichart. 2019. Deep contextualized self-training for low resource dependency parsing. *Transactions of the Association for Computational Linguistics*, 7:695–713. https://doi.org/10.1162/tacl_a_00294

Schlippe, Tim, ThuyLinh Nguyen, and Stephan Vogel. 2008. Diacritization as a machine translation and as a sequence labeling problem. In *Proceedings of the 8th Conference of the Association for Machine Translation in the Americas: Student Research Workshop*, pages 270–278.

Seddah, Djamé, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, et al. 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. https://doi.org/10.18653/v1/W13-4917

Spoustová, Drahomíra and Miroslav Spousta. 2010. Dependency parsing as a sequence labeling task. *The Prague Bulletin of Mathematical Linguistics*, 94:7. https://doi.org/10.2478/v10108-010-0017-3

Stanley, Richard P. 2015. *Catalan Numbers*. Cambridge University Press. https://doi.org/10.1017/CBO9781139871495

Strzyz, Michalina, David Vilares, and Carlos Gómez-Rodríguez. 2019a. Sequence labeling parsing by learning across representations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5350–5357. https://doi.org/10.18653/v1/P19-1531

Strzyz, Michalina, David Vilares, and Carlos Gómez-Rodríguez. 2019b. 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. https://doi.org/10.18653/v1/N19-1077

Strzyz, Michalina, David Vilares, and Carlos Gómez-Rodríguez. 2020. Bracketing encodings for 2-planar dependency parsing. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2472–2484. https://doi.org/10.18653/v1/2020.coling-main.223

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27.

Tenney, Ian, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, et al. 2018. What do you learn from context? Probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.

Tian, Yuanhe, Yan Song, Fei Xia, and Tong Zhang. 2020. Improving constituency parsing with span attention. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1691–1703. https://doi.org/10.18653/v1/2020.findings-emnlp.153

Vacareanu, Robert, George Caique Gouveia Barbosa, Marco A. Valenzuela-Escárcega, and Mihai Surdeanu. 2020. Parsing as tagging. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5225–5231.

Van Cranenburgh, Andreas, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111. https://doi.org/10.15398/jlm.v4i1.100

van der Goot, Rob, Ahmet Üstün, Alan Ramponi, Ibrahim Sharaf, and Barbara Plank. 2021. Massive choice, ample tasks (MaChAmp): A toolkit for multi-task learning in NLP. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 176–197. https://doi.org/10.18653/v1/2021.eacl-demos.22

Vaswani, Ashish, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237. https://doi.org/10.18653/v1/N16-1027

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

Vilares, David, Mostafa Abdou, and Anders Søgaard. 2019. Better, faster, stronger sequence tagging constituent parsers. 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 3372–3383. https://doi.org/10.18653/v1/N19-1341

Vilares, David 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. https://doi.org/10.18653/v1/2020.emnlp-main.221

Vilares, David, Michalina Strzyz, Anders Søgaard, and Carlos Gómez-Rodríguez. 2020. Parsing as pretraining. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9114–9121. https://doi.org/10.1609/aaai.v34i05.6446

Vinyals, Oriol, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. *Advances in Neural Information Processing Systems*, 28.

Wang, Bin, Jiangzhou Ju, Yang Fan, Xinyu Dai, Shujian Huang, and Jiajun Chen. 2022. Structure-unified M-tree coding solver for math word problem. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8122–8132. https://doi.org/10.18653/v1/2022.emnlp-main.556

Wang, Yufei, Mark Johnson, Stephen Wan, Yifang Sun, and Wei Wang. 2019. How to best use syntax in semantic role labelling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5338–5343. https://doi.org/10.18653/v1/P19-1529

Wang, Yiran and Masao Utiyama. 2024. To be continuous, or to be discrete, those are bits of questions. In *Proceedings of the 62nd Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers), pages 8036–8049. https://doi.org/10.18653/v1/2024.acl-long.436

Yang, Jie and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79. https://doi.org/10.18653/v1/P18-4013

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

Yang, Songlin and Kewei Tu. 2023. Don't parse, choose spans! Continuous and discontinuous constituency parsing via autoregressive span selection. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8420–8433. https://doi.org/10.18653/v1/2023.acl-long.469

Yu, Chen and Daniel Gildea. 2022. Sequence-to-sequence AMR parsing with ancestor information. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 571–577. https://doi.org/10.18653/v1/2022.acl-short.63

Zamaraeva, Olga and Carlos Gómez-Rodríguez. 2024. Revisiting supertagging for faster HPSG parsing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11359–11374. https://doi.org/10.18653/v1/2024.emnlp-main.635

Zhang, Yao zhong, Takuya Matsuzaki, and Jun'ichi Tsujii. 2009. HPSG supertagging: A sequence labeling view. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 210–213. https://doi.org/10.3115/1697236.1697277