CLEI electronic journal, Volume 25, Number 1, Paper 2, April 2022

Statistical Deep Parsing for Spanish: Abridged Version

Luis Chiruzzo, Dina Wonsever
Instituto de Computación, Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
{*luischir, wonsever*}*@fing.edu.uy*

Abstract

This document presents the development of a statistical HPSG parser for Spanish. HPSG is a deep linguistic formalism that combines syntactic and semantic information in the same representation, and is capable of elegantly modeling many linguistic phenomena. We describe the HPSG grammar adapted to Spanish we designed and the construction of our corpus. Then we present the different parsing algorithms we implemented for our corpus and grammar: a bottom-up strategy, a CKY with supertagger approach, and a LSTM top-down approach. We then show the experimental results obtained by our parsers compared among themselves and also to other external Spanish parsers for some global metrics and for some particular phenomena we wanted to test. The LSTM top-down approach was the strategy that obtained the best results on most of the metrics (for our parsers and external parsers as well), including constituency metrics (87.57 unlabeled F1, 82.06 labeled F1), dependency metrics (91.32 UAS, 88.96 LAS), and SRL (87.68 unlabeled, 80.66 labeled), and most of the particular phenomenon metrics such as clitics reduplication, relative referents detection and coordination chain identification.

Keywords: Parsing, HPSG, Spanish, Neural Parsing, LSTM

1 Introduction

Natural Language Processing is an interdisciplinary field that combines linguistics and artificial intelligence, its main intent is to create automatic methods to understand or generate human language. One historical approach to Natural Language Processing [1] proposes a possible set of linguistic knowledge that an NLP application should acquire, in increasing order of complexity: Phonetics and Phonology, Morphology, Syntax, Semantics, Pragmatics, and finally Discourse analysis.

This pipeline is designed so that earlier tasks extract more basic information from the text that could be used by later tasks, and in turn this would enable solving higher order NLP tasks such as automatic summarization, question answering or machine translation. It is clear that in such a scenario, improvements in each of the steps lead to further improvements in downstream tasks. This pipeline approach is nowadays gradually falling out of favor to more direct approaches like end-to-end systems that do not necessarily rely on explicitly annotated linguistic knowledge, but it is still widely used for many tasks, especially for languages with fewer linguistic resources than English. Besides seeking improvement in NLP tasks, the analysis of each one of the steps is in itself a very interesting problem that can shed light on important linguistic questions and, indirectly, give us insights on human language and the human mind itself.

In this work[1] we will mainly focus on the stage located in the middle of the pipeline: syntactic analysis. Syntax tries to model a language using a set of rules that govern how to build sentences. These sets of rules are known as **grammars**. The process of taking a sentence and building a syntactic representation of it is known as syntactic analysis, and it is generally referred to as **parsing**. A system that performs this kind of analysis is referred to as a **parser**.

There are several linguistic formalisms (grammars) that could be used to represent the syntax of a sentence. Some grammar formalisms are more shallow and just give surface information on the sentences and how their words interact, while others are deeper and often involve different types of syntactic and semantic information. One of these deep linguistic formalisms is Head-driven Phrase Structure Grammar

[1] This work presents an abridged version of the thesis *Statistical Deep Parsing for Spanish* [2].

(**HPSG**) [3], a rich linguistic formalism that combines syntactic and semantic information in its analyses and is able to model many interesting linguistic phenomena. We will focus on this formalism in this work.

Grammars and parsers can be developed with different intentions in mind. One approach is to build a grammar that is capable of correctly telling apart if a sentence is well-formed for a language or not: accepting or rejecting some sentences. However, natural language is written by people, which on the one hand are prone to make mistakes, and on the other hand are very creative so that new terms and expressions are coined everyday. There is another approach to grammar development and parsing that, instead of checking if a sentence is correct, tries to infer what is the most likely syntactic structure a sequence of words could have, so that sentences with small errors (e.g. typos or minor grammatical mistakes) might still get a good enough analysis that could be used by downstream applications. These systems rely on statistical knowledge of the language, generally obtained from a **corpus**, a collection of sentences that is used as a representation of the language. Corpora used for parser development should be large, comprehensive and varied enough so as to capture the main characteristics of the language we want to model.

Historically, English has been the most explored language in NLP. The amount of data, both annotated and unannotated, that exists for English is not comparable to what exists for other languages. Similarly, the state of the art performance for different NLP tasks in English is often much higher than for other languages. In this regard, HPSG is no exception: there are fast high performing parsers for English, but for other languages the research has in some ways lagged behind. In this work, we will try to further explore the adaptation of the HPSG deep linguistic formalism to Spanish.

It is our hope that this work could be able to narrow, even if very slightly, the existing gap between NLP capabilities for English and Spanish, and also that it could help foster the interest in developing and using resources for Spanish and for rich grammar formalisms.

1.1 Objectives

The objective of this work was to create a statistical HPSG parser for Spanish. In order to do this, the following milestones had to be completed:

- Define a HPSG grammar for Spanish that covers the language phenomena we want to address. We also wanted to make it flexible enough to be able to model many possible sentences, as we wanted to use it in the context of a statistical parser.
- Create a corpus of HPSG annotated sentences using that grammar. The parser would need to have a large collection of sentences in order to create good statistical models.
- Develop statistical parsing algorithms based on the information of the corpus. We focused mainly on applying neural network techniques to the problem of HPSG parsing, which have not yet been widely explored for this task.
- Analyze the performance of the parsing algorithms compared to other established Spanish baselines.

1.2 Contributions of this thesis

We created a simple yet powerful HPSG grammar for Spanish that models morphosyntactic information of words, syntactic combinatorial valence, and semantic argument structures in its lexical entries. The grammar uses thirteen very broad rules for attaching specifiers, complements, modifiers, clitics, relative clauses and punctuation symbols, and for modeling coordinations. In a simplification from standard HPSG, the only type of long range dependency we model is the relative clause that modifies a noun phrase, and we use semantic role labeling as our semantic representation.

We transformed the Spanish AnCora corpus using a semi-automatic process and analyzed it using our grammar implementation, creating a Spanish HPSG corpus of 517,237 words in 17,328 sentences (all of AnCora).

We implemented several statistical parsing algorithms and trained them over this corpus. The implemented strategies are: a bottom-up baseline using bi-lexical comparisons or a multilayer perceptron; a CKY approach that uses the results of a supertagger; and a top-down approach that encodes word sequences using a LSTM network.

We evaluated the performance of the implemented parsers and compared them with each other and against other existing Spanish parsers. Our LSTM top-down approach seems to be the best performing parser over our test data, obtaining the highest scores (compared to our strategies and also to external parsers) according to constituency metrics (87.57 unlabeled F1, 82.06 labeled F1), dependency metrics (91.32 UAS, 88.96 LAS), and SRL (87.68 unlabeled, 80.66 labeled), but we must take in consideration that the comparison against

the external parsers might be noisy due to the post-processing we needed to do in order to adapt them to our format. We also defined a set of metrics to evaluate the identification of some particular language phenomena, and the LSTM top-down parser outperformed the baselines in almost all of these metrics as well.

It is also possible to test the parsing strategies we implemented in the web site `parsur.com`[2]. The site lets you parse sentences using the CKY and the LSTM top-down strategies and provides an interactive visualization of the resulting HPSG trees.

2 Background

This section presents some basic information about the grammar formalism we use, HPSG, and some resources for that formalism that are relevant to our work. Then it describes the corpus we used as starting point for our work, which was not annotated in a HPSG format.

2.1 Head-driven Phrase Structure Grammars

Classic grammar formalisms like Context Free Grammars (CFG) focus on modeling the constituents of a sentence, and other formalisms more widely used nowadays like dependency formats focus on modeling the relations between words without considering constituents. There exist other types of grammars that take the best of both worlds, being strongly lexicalized and still focusing on modeling the constituency structure of the sentence. These grammar formalisms are generally called deep linguistic formalisms and they can deal with a wide range of linguistic phenomena and also could incorporate both syntactic and semantic information into their parse trees. Being strongly lexicalized, most of the complexity of the grammars is encoded in the description of its lexical entries, so these formalisms often have very few rules and the rules can generally be applied in a broad range of scenarios, but only when they are licensed by the lexical entries. Examples of these formalisms include Combinatory Categorial Grammar [4], Tree Adjoining Grammar [5], Lexical Functional Grammar [6] and Head-driven Phrase Structure Grammar [3]. These deep grammars differ in the way they describe the lexical entries, how they can be combined to form greater units, and the shape of the structure that represents a parse tree in each grammar.

Head-driven Phrase Structure Grammar (HPSG) is a strongly lexicalized grammar formalism based on feature structures with a unification operation. Each lexical entry is represented with a **feature structure**, i.e., an association of features and values that describes the behavior of the word. Feature structures can be combined using **unification**: a process by which the features of two feature structures are merged forming a new one containing the features of both, and goes on recursively unifying the values when the same feature is found in both structures.

The rules of the grammar and eventually the whole parse trees are also represented as feature structures. Every rule defines one of its daughters as the syntactic head of the rule, and there are a series of **principles** that indicate the way the daughters features are percolated to the parent structure. The grammar is head-driven because the principles indicate that, unless explicitly specified, the parent structure will inherit the features of the daughter marked as head.

Because much of the combinatorial information is encoded in the lexical entries and not in the rules, there tend to be very few rules in an HPSG grammar compared to other formalisms like CFG. These are generally very broad rules, inspired mainly in X' theory [7], for example describing the way to attach a specifier, a modifier or a complement to a head.

Consider the following sentence sample:

(1) *La gata negra duerme (The black cat sleeps)*

Figure 1 shows a possible tree for sentence 1 using our HPSG grammar. Notice that each word defines the valence feature it expects, and in the tree these features are coindexed with the corresponding word. For example, the noun *"gata"* indicates it expects a determiner as specifier, and in the tree the `SPEC` feature points to structure 2, which is the determiner *"la"*, so we call these representations reentrant trees.

HPSG grammars are also very good at modeling other aspects of the language such as the subcategorization of verbs, as each verb can be modeled with a lexical entry that specifies exactly the types of complements and subject it expects and, as we will see, it can also link this information with a semantic representation.

2.2 Spanish Resource Grammar

The only antecedent to our work in terms of Spanish implementations of HPSG is the Spanish Resource Grammar (SRG) [8]. It contains more than 50,000 lexical entries, 64 lexical rules and 191 combinatorial

[2] `http://parsur.com/`

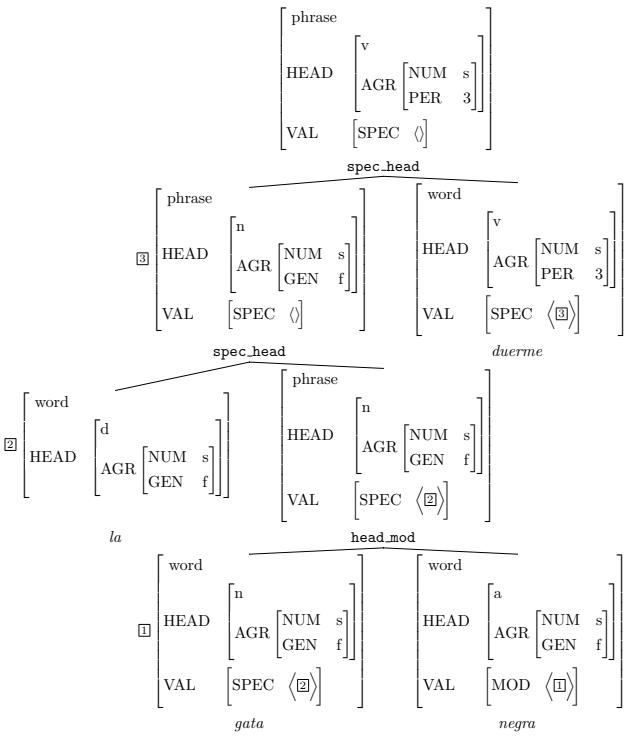

Figure 1: HPSG tree for *"La gata negra duerme"*.

rules for forming phrases. The rules were developed manually and they used manual and semi-automatic processes to build the lexicon [9]. The parse trees obtained using SRG are very rich trees that support many of the linguistic constructions provided by the HPSG theory.

Due to the use of the LKB parser, the original version of this grammar did not include statistical analysis support, for example for tree disambiguation. However, later on a disambiguation post-processing was added [10]. In this work we focus on a different approach for building a Spanish HPSG grammar: beginning with an annotated corpus, and using this information to extract the lexical entries and perform a statistical modeling of the grammar rules and lexical entries. This approach might be more flexible, but we must take into account that it will rely strongly on the content of the original corpus and how well it represents the language.

2.3 Enju

The grammar development strategy we follow in this work is inspired by the Enju system[3] [11], a statistical HPSG parser with high coverage for the English language. Enju is a high performance parser that also successfully resolves some complex linguistic phenomena like raising verbs and control verbs, and also has some success at analyzing coordinated phrases. It also has versions for Japanese and Chinese.

For building this parser, they followed a radically different approach than the one used for the SRG. Instead of manually crafting the grammar rules and lexical entries, they created an HPSG corpus based on the Penn Treebank corpus [12]. As the syntactic annotations of this corpus are not directly comaptible with an HPSG grammar, they implemented a conversion process that adapts its rules to a format similar to HPSG [13]. Then they used the resulting corpus to build the lexical entries and extract the rule application probabilities.

Unlike the English Resource Grammar, the grammar used by the Enju parser incorporates some more elements of the HPSG version in [3]. Their approach for semantics is also different than SRG. While SRG uses Minimal Recursion Semantics, the semantic representationin Enju is more akin to standard Semantic Role Labeling using PropBank notation [14], which is also the strategy we used in our implementation. This is a simpler approach that lacks the compositional restrictions and more complex modeling found in MRS.

Enju's approach of transforming a constituency treebank into HPSG and then training a parser was also followed more recently in [15], which combined the constituency and dependency versions of the Penn Treebank to infer HPSG annotations. Their parser obtains very good performance over the Penn Treebank,

[3] `https://mynlp.is.s.u-tokyo.ac.jp/enju/index.html`

being the state of the art for constituency parsing in English (96.33 F1 score), which might indicate that besides modeling rich linguistic structures, HPSG is also able to attain high parsing performance.

2.4 AnCora

AnCora [16] is a corpus of Spanish and Catalan texts of approximately 500,000 words. It contains 17,000 sentences in around 1,600 news articles. All sentences are annotated syntactically in a CFG-style format. Words and constituents also incorporate more information as XML attributes, for example:

- Morphological attributes: The parts of speech and morphological information of the words are encoded using the EAGLES tagset[4].
- Grammatical categories of constituents: There is an extensive set of categories including `grup.verb` for verbal periphrases, `S` for subordinate sentences, `sn` for noun phrases and `grup.nom` for noun groups (noun phrases that lack a specifier). However, the annotation is not completely consistent across the whole corpus.
- Semantic argument structure: They use the `arg` attribute for marking semantic roles using the PropBank notation.
- Verbal subcategorization: Encoded in the lexical semantic structure (`lss`) attribute with thousands of different subcategorization frames.
- Null subjects: Instances of null subjects in the corpus, which are very common in Spanish, are marked as a noun phrase `sn` with an attribute `elliptic='yes'`. This is useful as this structure is marked as subject and has the corresponding semantic role when available.

Using these rich annotations, it is possible to create rules that transform the structures in this corpus into other formats like the developers of Enju did for the Penn Treebank. However, compared to Penn Treebank, the structure of AnCora is more complex in terms of categories and in the level of elements nesting (for example, a noun phrase `sn` usually will contain a nominal group `grup.nom` and optionally a specifier `spec`, and each structure tends to contain more nested structures), although the annotation is not consistent in some parts of the corpus. There is also a great number of rules per category and a huge diversity in the way the rules are used. For example, in the corpus there are 5,826 rules for subordinate sentences (`S`), 2,403 for sentences (`sentence`), 905 for nominal groups (`grup.nom`) and 295 for noun phrases (`sn`).

3 Grammar

This section describes the grammar we use: our version of HPSG adapted to Spanish. In HPSG grammars, both lexical entries and rules are defined as feature structures, and their constraints (selection constrains or rule application constraints) are specified in their features. We will describe the general feature structure for expressions (both words and phrases) and present the different grammar rules that are used to combine the expressions and form the parse trees. An initial version of this grammar and the corpus can be found in [17].

3.1 Feature structure for expressions

The feature structure we use for a word or phrase contains the features shown in figure 2. Notice that many of these features are implemented as lists of expressions but in some cases the only possibilities for the list are only zero or one expressions.

Syntactic features

- `SPEC`: List of expressions that are expected as specifiers of a node. In general there should only be zero or one of these. As in [18], the specifier feature is used to model both the specifiers of nouns (generally determiners) and the subjects of verbs.
- `COMP`: List of expressions that are expected as complements of a node. This list is unbounded, as some words (especially verbs) can have multiple complements, though in general it has up to four or five elements.

[4] `http://blade10.cs.upc.edu/freeling-old/doc/tagsets/tagset-es.html`

$$
\begin{bmatrix}
\text{word} & \\
\text{HEAD} & \begin{bmatrix} \text{n} \mid \text{v} \mid \text{a} \mid \ldots & \\ \text{AGR} & \begin{bmatrix} \text{PER} & 1 \mid 2 \mid 3 \\ \text{NUM} & \text{sg} \mid \text{pl} \\ \text{GEN} & \text{f} \mid \text{m} \\ \ldots & \end{bmatrix} \end{bmatrix} \\
\text{VAL} & \begin{bmatrix} \text{SPEC} & \langle \text{expr} \rangle \\ \text{COMP} & \langle \text{expr} \rangle \\ \text{CLITIC} & \langle \text{expr} \rangle \\ \text{MOD} & \langle \text{expr} \rangle \\ \text{LEFT\_COORD} & \langle \text{expr} \rangle \\ \text{RIGHT\_COORD} & \langle \text{expr} \rangle \end{bmatrix} \\
\text{NONLOCAL} & \begin{bmatrix} \text{REL} & \langle \text{expr} \rangle \end{bmatrix} \\
\text{SEM} & \begin{bmatrix} \text{ARG0} & \langle \text{expr} \rangle \\ \text{ARG1} & \langle \text{expr} \rangle \\ \ldots & \\ \text{IS\_ARG0} & \langle \text{expr} \rangle \\ \text{IS\_ARG1} & \langle \text{expr} \rangle \\ \ldots & \end{bmatrix} \\
\text{TEXT} & \text{text}
\end{bmatrix}
$$

Figure 2: Feature structure for a word.

- `MOD`: List of expressions that this node is expected to modify. There can be only zero or one elements in this list.
- `CLITIC`: List of expressions that are expected as clitics of a node. Because of Spanish constraints, this list should have up to two elements.
- `LEFT_COORD` (and `RIGHT_COORD`): List of expressions this node is expected to have on its left (or right) in order to form a coordination (only zero or one expressions).
- `REL`: List of expressions that this node is expected to be modifying as a relative clause. There can be only zero or one elements in this list.

Semantic features Inside the `SEM` feature, there are features for determining the semantic role label structure of the expressions that are attached to a word. Unlike the original HPSG grammars, we use a representation based on the PropBank [14] notation of semantic arguments. The features we use are the following:

- `ARG0`, `ARG1`, `ARG2`, `ARG3`, `ARG4`, `ARGM` and `ARGL`: The PropBank features that are included as annotations in AnCora. `ARG0` indicates a proto-agent, `ARG1` indicates a proto-patient, while the rest of the numeric arguments indicate roles that depend on the verb. `ARGM` corresponds to adjuncts and `ARGL` indicates arguments of light verbs [16].
- `ARGC`: This is a special feature we use to model verbal complements for verb phrases, used together with the semantic complement rule (see section 3.2).
- `IS_ARG0`, `IS_ARG1`, `IS_ARG2`, `IS_ARG3`, `IS_ARG4`, `IS_ARGM` and `IS_ARGL`: These are the symmetric features for modeling arguments that are introduced by modifiers.

3.2 Rules

Our grammar uses only thirteen very generic rules. Most the rules have left and right versions for attaching a dependent to the left or to the right of a head. The rules are the following:

- Specifier rules: Two rules for attaching a specifier to the left or to the right of a head (`spec_head` and `head_spec`, see figure 3a). The `head_spec` rule for attaching is our way of modeling the phenomenon of postponed subject in Spanish, where the subject is written on the right of the verb instead of the more usual SVO position, for example in *"Llego el tren"* (*"The train arrived"*).
- Complement rules: Two rules for attaching a complement to the left or to the right of a head (`comp_head` and `head_comp`).

- Semantic complement rule: One rule for forming verb phrases were the head keeps the morphological features and the complement keeps the arguments structure (`head_comp_sem`).
- Modifier rules: Two rules for attaching a modifier to the left or to the right of a head (`mod_head` and `head_mod`, see figure 3b).
- Clitics rule: One rule for attaching a clitic pronoun to the left of a verb (`clitic_head`). The clitic reduplication phenomenon [19] is also modeled using a coindexation of semantic structure between the clitic and the real argument, for example in *"Yo lo vi a Juan"* (*"I saw Juan"*).
- Relatives rule: One rule for attaching a relative phrase that acts as a modifier of a noun (`head_rel`). This is a simplification from other version of HPSG in that it is the only type long range dependency we are modeling in our grammar. We keep the link between the head of the relative phrase and the noun it is pointing to, such as in *"El libro que Juan compró"* (*"The book Juan bought"*).
- Punctuation rules: Two rules for attaching a punctuation symbol to the left or to the right of a head (`punct_head` and `head_punct`).
- Coordination rules: Two rules for binarizing coordinations (`coord_left` and `coord_right`).

$$\boxed{1} \; + \; (\mathrm{H}) \begin{bmatrix} \text{expr} & \\ \text{VAL} & \left[\text{SPEC} \quad \left\langle \boxed{1} \right\rangle\right] \end{bmatrix} \rightarrow \begin{bmatrix} \text{phrase} & \\ \text{VAL} & \left[\text{SPEC} \quad \langle\rangle\right] \end{bmatrix}$$

(a) `spec_head`

$$(\mathrm{H}) \boxed{1} \; + \; \begin{bmatrix} \text{expr} & \\ \text{VAL} & \left[\text{MOD} \quad \left\langle \boxed{1} \right\rangle\right] \end{bmatrix} \rightarrow \boxed{1}$$

(b) `head_mod`

Figure 3: Examples of HPSG rules in our grammar: rule for attaching a specifier to the left of a head, and for attaching a modifier to the right of a head.

4 Description of the Corpus

The corpus we built is based on the AnCora corpus (see section 2.4) and was transformed using a set of heuristics and later analyzed using a feature structure implementation of our grammar.

The transformation process comprises a top-down step followed by a bottom-up step, with the aim of transforming all sentences in AnCora into **elementary HPSG trees**, i.e. a head surrounded by its direct dependents (complements, specifier, modifiers). The top-down process has the aim of breaking down complex structures found in AnCora, which tends to have a rather flat structure. The bottom-up process is in charge of binarization of the structures using a set of hand crafted rules for head-finding inside the constituents, and other heuristics for detecting the syntactic rule to apply at each step.

This initial transformation process is described in a previous work [20]. In this work, we needed to expand and improve this transformation with the intention of modeling several complex structures we found in the corpus, which needed to be handled with special care. One of these complex structures that are modeled a differently than the rest of the components in AnCora are the verb phrases. Verb phrases were manually analyzed and transformed using special rules into verb structures that use the `head_comp_sem` rule to appropriately model the morphological and argumental features of the structures.

Other structures that needed special attention were the application of clitics, the detection of relative clauses that act as noun modifiers and the use of null subjects.

The final corpus has approximately half a million words (517,237) in over 17,000 sentences. Table 1 shows the number of lexical frames (superclass of lexical entries without considering the word and the morphological features), lexical entries and instances discriminated by part of speech.

We split the contents of the corpus in training, development and test partitions according to the standard AnCora partitions used in CoNLL-2009 shared task [21]: around 80% words for training, 10% for development and 10% for test. Table 2 shows the number of documents, sentences and words in each partition.

The number of times each rule is applied is shown in table 3.

POS	Frames	Entries	Instances
a - adjective	120	11,697	35,939
c - conjunction	191	796	27,067
d - determiner	11	303	76,135
f - punctuation	57	136	65,546
i - interjection	8	62	99
n - noun	226	34,193	121,113
p - pronoun	38	431	22,692
r - adverb	74	2,214	18,952
s - preposition	227	1,841	79,901
v - verb	2,234	28,486	61,699
w - date	21	1,043	2,731
z - number	38	2,392	5,363
Total	3,245	83,594	517,237

Table 1: Number of lexical frames, entries and instances for each part of speech.

	Train	Dev	Test	Total
Documents	1,312	155	168	1,635
Sentences	14,018	1,638	1,692	17,348
Tokens	418,319	49,338	49,580	517,237

Table 2: Statistics of the corpus partitions in terms of number of documents, sentences and tokens.

Rule	Train	Dev	Test	Total
`clitic_head`	6,554	789	727	8,070
`comp_head`	4,679	544	553	5,776
`coord_left`	14,725	1,725	1,785	18,235
`coord_right`	14,725	1,725	1,785	18,235
`head_comp`	116,057	13,597	13,568	143,222
`head_comp_sem`	8,325	935	1,002	10,262
`head_mod`	74,171	8,979	8,825	91,975
`head_punct`	31,266	3,767	3,684	38,717
`head_rel`	6,269	731	737	7,737
`head_spec`	5,272	599	606	6,477
`mod_head`	22,520	2,662	2,691	27,873
`punct_head`	18,424	2,192	2,237	22,853
`spec_head`	89,964	10,501	10,742	111,207

Table 3: Number of times each rule is applied in the corpus for each corpus partition.

5 Parser Development

In this section describe the different parsing strategies we tried for building our parser: a bottom-up parsing baseline, a CKY approach, and a top-down approach.

5.1 Bottom-up strategy

To begin with our parsing experiments, we first designed a very simple parsing strategy that we use as a baseline in order to compare to more complex approaches. This simplest strategy uses a greedy approach to build the parse tree in a bottom-up way. Consider the following example:

(2) *El niño come una manzana roja (The kid eats a red apple)*

The expected constituency tree for sentence 2 in our grammar is shown in figure 6.

Given the way the grammar works, we can be sure that in the expected tree there are at least two consecutive words of the original sentence that should be combined together to form a phrase. The aim of the bottom-up baseline parser is to find a suitable pair of consecutive words to combine, select the appropriate rule, and substitute the words for a phrase. Then it will proceed iteratively until all words are structured into phrases. Let us see how it works in this example.

First of all, there are two possible phrases that can be extracted in this case:

[El niño] come una [manzana roja]

If the process extracts any of those two phrases, the final tree will be the same. The process might decide to extract, for example, the phrase *[manzana roja]* and decide that the rule to apply is `head_mod`. Then it will substitute the pair of words for the syntactic head of the phrase, leaving the following:

El niño come una manzana

Now there are two more options to reduce: *[El niño]* and *[una manzana]*. The process might decide to extract *[El niño]* and label it with the `spec_head` rule. After reducing the phrase, the rest of the sentence looks like the following:

niño come una manzana

From now on, there is only one way of extracting phrases and reducing the tree in order to get the gold parse tree. The process must first extract *[una manzana]* as `spec_head` and substitute it with *manzana*, then it must extract *[come manzana]* as `head_comp` (leaving *come*) and finally extract *[niño come]* as `spec_head`. With this sequence of reductions, the process yields the parse tree shown in figure **??**.

What the process needs to learn is how to decide which pair of words is the most suitable to extract on the next step, and which rule it should apply. We designed and tested several scoring functions for deciding which words to combine in increasing order of complexity: approaches based on counting the number of times each pair of words appears in the corpus of tree derivations (bilexical counts), and approaches based on training a multilayer perceptron neural network for predicting the best pair of words and the corresponding rule to apply. The best scoring function we found uses a two hidden layers of size 250 and 150 and the inputs are 300-dimensional word embeddings trained over a corpus of six billion words in Spanish.

For the sake of completeness, this bottom-up strategy also uses a very simple baseline heuristic for calculating SRL: coindex the `SPEC` feature of a predicate with `ARG0` and the `COMP` feature to `ARG1`.

5.2 CKY with Supertags

The bottom-up process described so far has some very important drawbacks. First of all, once the next pair of words to reduce has been decided, there is no turning back to analyze other options. That is because the process is greedy. This has the advantage of being fast, as each option is only considered once, but also the disadvantage that errors committed early in the process condition the rest of the parsing process and could easily cascade into other errors.

Another problem with the approaches above is that they only consider the information conveyed by words, but not other information that would be available in a HPSG approach. Furthermore, once we decided which pair of words to reduce, we simply substitute the pair for the head word according to the selected rule. This implies forgetting the derivation history so far, information that could be exploited in order to improve the rest of the process.

In order to deal with these problems, we implemented a more standard strategy that has been used for parsing HPSG [22] and other grammar formalisms. Given our grammar is binarized, it is possible to use the well known CKY algorithm for parsing [23–25]. Instead of using a greedy process, CKY is a dynamic programming algorithm that keeps a chart of partially parsed trees and analyzes all possible combinations bottom-up in polynomial time. It is possible that many valid trees are generated in the process, so we also need a probabilistic model for determining which of the trees is the most likely one.

The simplest way of implementing this CKY process would be directly using the unification process: at each step we try to apply every rule (unifying the rule with the left and right subtrees), and we add the tree to the chart if the unification is sound. This has an important shortcoming: the unification process is so slow that parsing with this strategy becomes impossible. So in order to make this work in a reasonable time, we implemented a speed up strategy based on the use of supertags: We designed a set of supertags based on the grammar categories that contain the necessary information to infer the possible rules to apply given a pair of supertags.

Supertags are a technique for creating rich categories for parts of speech that contain higher order combinatorial information that could be used for parsing deep grammars. First defined for parsing with TAG [26] grammars, they have been also applied to parsing CCG [27] or HPSG [22, 28] grammars. Furthermore, a preliminary version of our HPSG supertags is described in [29]. The task of supertagging is an extension of POS-tagging that implies tagging with these rich categories instead of the simpler parts of speech.

The supertags are used in two ways in our process: on the one hand we implemented a standard approach to supertagging meant to disambiguate the possible categories to apply to a word, and on the other hand we will use the same tags to speed up the calculation of the possible rules to apply.

Our definition of supertags tries to represent the way a word is being used in the context of a sentence. It describes the feature slots the word has to fill and in which positions with respect to the current word.

These supertags are designed to be very expressive in terms of the description of the word, so the parsing process becomes as unambiguous as possible.

Consider again the sample sentence 2:

El niño come una manzana roja.
(The kid eats a red apple.)

The main verb of the sentence (*come*) has a noun phrase specifier to its left (*El niño*) and a noun phrase complement to its right (*una manzana roja*). This will be the information conveyed by the supertag for *come*, as shown below:

come/`v-sna0-x-cna1`

The leftmost character is the part of speech of the word. The rest of the supertag is a description of the use of the word in that context, where `x` represents the position of the word. In this case, `sna0-x-cna1` encodes the following information: the word will have a noun phrase acting as specifier on its left (`sn`) coindexed with `ARG0`, plus a noun phrase acting as a complement on its right (`cn`) coindexed with `ARG1`.

The corresponding supertags for the whole sentence are the following:

El/`d-x` *niño*/`n-sd-x` *come*/`v-sna0-x-cna1` *una*/`d-x` *manzana*/`n-sd-x` *roja*/`a-mn-x` ./`f-x`

The CKY algorithm relies on knowing the exact categories of the words to find the valid rules to apply, but when parsing a sentence from scratch that information would not be available. As the number of possible categories per word is large, we trained a supertagger for calculating the most suitable supertags given a sentence. The supertagger uses as information the words and POS-tags for the sentence and returns a sequence of supertags.

This supertagger is built using stacked LSTM neural networks. The structure of the network (after tuning against the development corpus) is the following (see figure 4):

- Input: Words and corresponding POS-tags
- Embeddings layer: word embeddings of size 300, POS embeddings of size 5.
- LSTM layers: Three layers of stacked bi-directional LSTMs with size 450 in each direction and activation `tanh`.
- Dense layer: A fully connected layer of size 300 and activation `tanh`.
- Output: Layer that selects one out of 4,146 possible supertags with activation `softmax`.

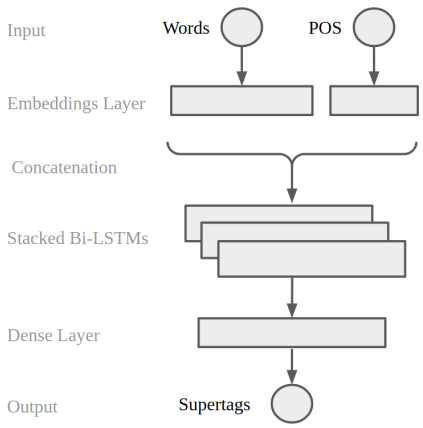

Figure 4: Architecture of the neural network for supertagging.

Using this network, we get a 89.1% accuracy over the development corpus for the top selected tag, 94.3% when choosing the top two tags, and 96.0% when choosing the top three tags.

We implemented the CKY algorithm adapted to our grammar rules and our probabilistic model. The number of possible rule combinations grows very steeply with the size of input, and this makes the parsing process very time consuming for long sentences. Because of this, we limit the number of derivations to

explore in each cell of the CKY matrix to a maximum of n, taking only the partial derivations that yield the highest probability for the same combination of words. Initial experiments over the development corpus indicated that the best combination of speed and performance could be achieved using $n = 5$, so we used that number in the rest of our experiments.

However, as the performance of the supertagger is not perfect, the sequence of supertags selected might be invalid for forming a tree according to the grammar rules. To mitigate this problem, if a tree is not found we enable two fallback rules with very low probability (`head_none` and `none_head`) that combine two arbitrary nodes and take either the left one or the right one as heads. These rules guarantee that a tree will be found, but they make the process much slower as many more subtrees are tried during parsing. When the fallback rules are enabled, we use $n = 2$ instead of $n = 5$ so the process becomes faster. As the difference between using the tags predicted by the supertagger and the gold supertags is so large, we will present results for both approaches, considering the gold supertags method an upper bound of performance to our CKY process.

5.3 Top-down strategy

The top-down parsing strategy takes a completely different approach. This strategy will try to leverage information from all the text sequence instead of the more localist approaches seen so far. Initial experiments with this architecture were reported in [30]. In this case, the parsing process consists in a series of steps that incrementally build the parse tree: splitting a sentence into a binary tree, finding out the rules that apply to the nodes of the tree, and finally determining what nodes should be labeled with semantic argument categories.

Let us walk through the proposed parsing process using again the sample sentence 2 *"El niño come una manzana roja"*.

Step 1: Splitter In the first stage, the process will take the whole sentence and split it in two sequences of words. The resulting subsequences are expected to be constituents of the sentence. In this case it should split the subject and the predicate. The result will look like the following:

[*El niño*] [*come una manzana roja*]

This process is repeated for each subsequence with three or more words, as sequences with two words are trivially split and sequences with only one word are already leaves in the tree. In this case, [*El niño*] has two words, so only the second subsequence will be split. The process should separate the verb from the object, resulting in the following:

[*come*] [*una manzana roja*]

Now the first subsequence has only one word (it is a leaf), and the process continues with the second subsequence separating the determiner from the rest of the noun phrase:

[*una*] [*manzana roja*]

At this point, there are no more non-trivial sequences to split, so the original sentences has been effectively transformed into a binary tree of words. The binary tree after step 1 for this example is shown in figure 5.

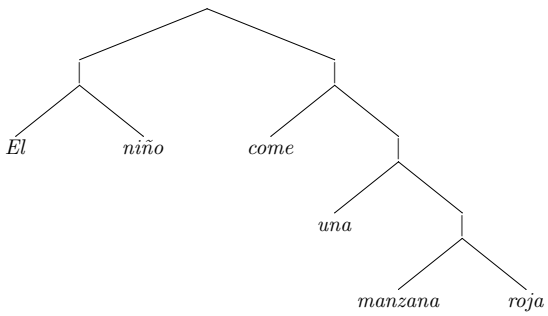

Figure 5: Unlabeled constituency tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*), built after step 1.

Step 2: Rules After the tree is created, the second stage transverses all pairs of branches and tries to find the most suitable rule that describes the relation between those branches. For example:

$$[\ \textit{El}\ ]\ [\ \textit{niño}\ ] \rightarrow \texttt{spec\_head}$$

$$[\ \textit{come}\ ]\ [\ \textit{una manzana roja}\ ] \rightarrow \texttt{head\_comp}$$

$$[\ \textit{manzana}\ ]\ [\ \textit{roja}\ ] \rightarrow \texttt{head\_mod}$$

After this stage is completed, the tree looks as the one shown in figure 6.

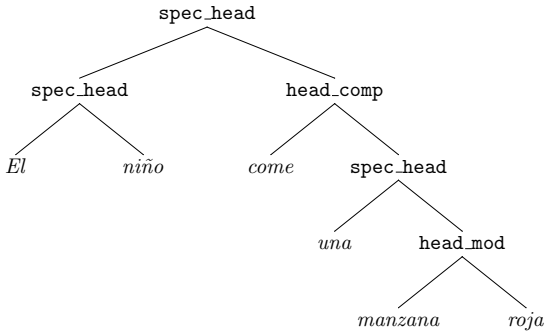

Figure 6: Labeled constituency tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*).

Step 3: Arguments Once the tree is created and the syntactic features are in place, the third step tries to determine which of the syntactic arguments of the predicates (in our case all verbs, nouns and adjectives) should also be set as semantic arguments.

Given the rules determined after step 2 for each pair of branches in the tree, we can infer the head for each constituent. This third step considers each head and each target argument that belongs to that head. In our example:

$$[\ \underline{\textit{El niño}}_{SPEC}\ \underline{\textit{come}}_{HEAD}\ \textit{una manzana roja}\ ] \rightarrow \texttt{arg0}$$

$$[\ \textit{El niño}\ \underline{\textit{come}}_{HEAD}\ \underline{\textit{una manzana roja}}_{COMP}\ ] \rightarrow \texttt{arg1}$$

$$[\ \textit{una}\ \underline{\textit{manzana}}_{HEAD}\ \underline{\textit{roja}}_{MOD}\ ] \rightarrow \texttt{none}$$

A simplified version of the final tree for our example, after the three stages have been completed, is shown in figure 7. This final version of the tree contains the information about the arguments of each predicate and the semantic roles they have according to the argument structure.

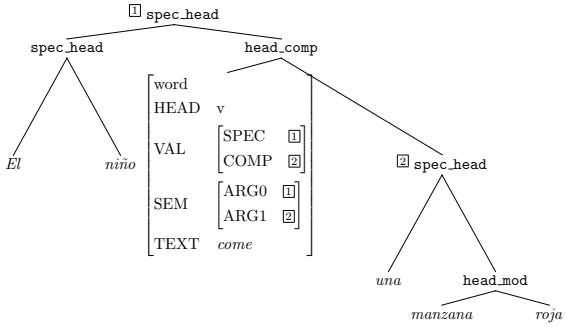

Figure 7: Simplified tree for *"El niño come una manzana roja"* (*"The kid eats a red apple"*) after step 3.

Notice that this top-down strategy does not use any subcategorization information for the lexical entries, for example it does not use the supertags like the CKY strategy does. Instead, this process just the uses the words of the sentence as input, encoded as word embeddings.

We experimented with several neural network architectures for fulfilling the three steps of the process. The first experiments used three different networks that were in charge of each of the different steps. Then we tried unifying steps one and two (calculating the best split point and predicting the correct rule), and using another network for predicting the semantic arguments. Finally we tried unifying the three steps into one network. The results of these experiments over the development set showed that the best strategy in our case is the second one: one network for calculating split and syntactic rule, and one network for semantic arguments.

Steps 1 and 2: Split-Rule model This model has an architecture similar to the supertagger described in section 5.2. It takes as input each word of the sentence at a time and returns the split probability for each word, and also the probability of rule application for each possible split point. The core of this model is a three-layered stacked LSTM network that transverses the sentence and returns, for each word, the probability of it to be the split boundary and the most likely rule to apply if that split point is used.

The output of the network is in fact a `softmax` over all the possible rules plus the `none` rule for indicating that the word should not be used as a split point. After a sequence is processed, we first look at the word with the minimum score for the `none` output, which is selected as split point. Then we look at the corresponding rule probabilities for that point and select the most likely rule.

The structure of the network is the following (see figure 8):

- Input: Sequence of words.
- Embeddings layer: Word embeddings of size 300.
- LSTM layers: Three layers of stacked bidirectional LSTMs.
- Dense layer: A fully connected layer.
- Output: Split probability and rule to use for each word.

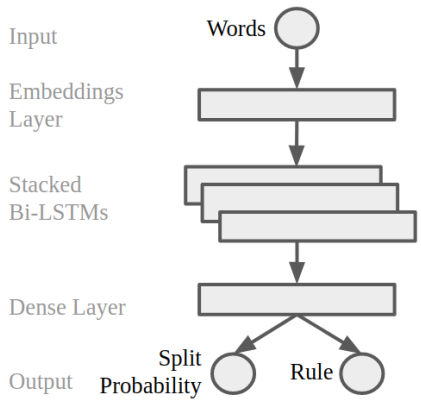

Figure 8: Architecture of the neural network for merged steps 1 and 2.

Step 3: Argument model The network for this model is more complex. In this case the network will take two sequences of words which are contained inside each other: one of them is the argument to classify, and the other is the largest constituent that contains the argument. It also takes the head of the constituent and the grammar rule that relates the head to the argument. The result of the network is the semantic role label that should be applied for this argument (or `none`).

The core of the model is a stack of three bidirectional LSTM layers, both input sequences (the argument and the other constituent) are run through the LSTM layers, and after that their result is concatenated together with the head and grammar rule information before calculating the final output.

This network is executed only for possible arguments of predicates, i.e. it is used for each verbal, nominal or adjectival head and all of their dependents. It is trained in the same way: using instances of <head, dependent> pairs from the training corpus with verb, noun or adjective heads, whether their dependent is a semantic argument or not.

The structure of the network (as shown in figure 9), is the following:

- Input: Structured input with one word (the head, which is the predicate), two sequences of words (the whole constituent, and the target argument), and the syntactic valence of the target argument.
- Embeddings layer: Word embeddings of size 300. The embeddings layer is used for transforming the head and the two sequences of words.
- LSTM layers: Three layers of stacked bidirectional LSTMs.
- Dense layer: A fully connected layer. It takes as input the concatenation of the result of all the previous layers.
- Output: Label indicating the arg type (`none`, `arg0`, `arg1`, `arg2`, `arg3`, `arg4`, `argm`, `argl`, `argc`).

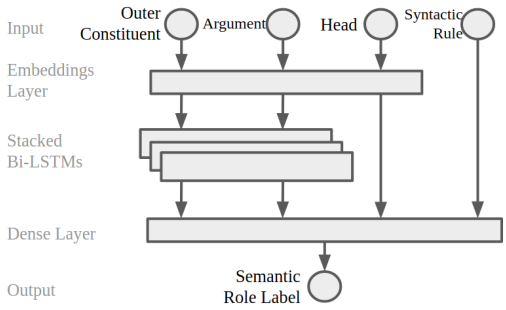

Figure 9: Architecture of the neural network for step 3.

5.4 Training details

The following applies to all the neural network models we trained. The models were implemented in the `keras` library [31] over `tensorflow` [32]. The word embeddings layer uses the collection described in section 5.1. It is an embeddings collection of 1,146,242 vectors of dimension 300, and we created an unknown token for each POS tag (considering morphological information) for handling out of vocabulary tokens.

We tuned several hyperparameters against the development corpus, but we are only reporting the results of the best performing model for each set of experiments. We applied dropout to LSTM and dense layers, in general varying between 0.1 and 0.5. We trained several configurations for every experiment varying the number of units (between 100 and 600 units) and the activation functions (generally `relu` or `tanh`) for each layer. When using a stack of LSTM layers, we varied between one and three layers. For the bottom-up baseline MLP architectures, we varied between one and two dense layers.

We also used the early stopping technique during training, with a held-out set of between 10% and 20% of each training set, depending on the experiment.

6 Evaluation

In this section we show the experimental results for our parsing strategies over the test corpus and compare them with other Spanish parsers in terms of syntactic parsing and semantic role labeling. We will also make a comparison of execution times and an analysis of the performance for some particular phenomena. An evaluation of some of the parsing approaches and their comparison can be found in [33].

6.1 Evaluated systems

We compare the strategies defined in section 5 with other six well established baselines for Spanish parsing. Unfortunately, none of the external baselines works with a formalism similar to the one we use, so we will only be able to compare some of the metrics for them. The only other parser that would use a similar structure that we know of (the Spanish Resource Grammar) is not available for comparison. It also has problems for parsing longer sentences, and there are many of these in the corpus.

- Bottom-up baselines: These are the approaches presented in section 5.1, both using simple counts of words or using multi-layer perceptrons for considering more context. These are the systems **Bottom-up Bilex Context 0**, **Bottom-up Bilex Context 1**, **Bottom-up MLP Context 1** and **Bottom-up MLP Context 2**.

- CKY approaches: We consider two approaches based on CKY, as presented in section 5.2. One of them is the unrealistic model that starts from the gold supertags, which we consider is an upper bound for the CKY process (**CKY Gold tags**). The other one is the more realistic CKY process that first uses a supertagger to predict the lexical frames it will have to use, and it will forcefully perform worse than the previous one (system **CKY Supertagger**).

- LSTM top-down system: This is the approach described in section 5.3, which uses a greedy top-down process with a LSTM for finding the structure and rules, and a second neural network for calculating the semantic arguments (system **LSTM Top-down**).

- FreeLing: FreeLing [34] is an important suite of NLP tools developed by Universitat Politècnica de Catalunya that has several Spanish models which include both dependency parsing and SRL. The ones we used for this comparison are:

  - **FreeLing Txala** [35, 36] is a dependency parser based on transforming the output of a previous rule-based parser into dependency format.
  - **FreeLing Treeler**[5] is a parser that uses a factorization or decomposition model trained using several statistical models (log linear, max-margin and perceptron).
  - **FreeLing LSTM** is a LSTM implementation of a dependency parser trained over AnCora.

- spaCy: spaCy[6] is a suite of NLP tools developed by Explosion AI. It contains models for several languages and in particular two dependency parsers and named entity recognizers for Spanish. These models are convolutional neural networks trained with multi-task training over the Universal Dependencies conversion of AnCora and the WikiNER corpus. We use models `es_core_news_sm` (small) and `es_core_news_md` (medium), the large model was not available. These models perform dependency parsing but not SRL.

- UDPipe: UDPipe [37] is a trainable pipeline that provides tokenization, morphological analysis and dependency parsing using the Universal Dependencies format. It was used as base implementation in CoNLL competitions. The parser uses a transition-based algorithm implemented with a simple neural network with one hidden layer, it is very fast and robust. They provide pre-trained models for several languages. We are using model `Spanish-AnCora`, which was trained with the Universal Dependencies conversion of AnCora. Like the spaCy models, this parser returns dependency parsing but not SRL.

6.2 Global metrics

The global metrics we used are the following: Unlabeled Constituency F1 (**U-Cons**), Labeled Constituency F1 **L-Cons**, Unlabeled Dependency Accuracy (**L-Dep**) analogous to dependency parsing Unlabeled Attachment Score (UAS), Labeled Dependency Accuracy (**U-Dep**) analogous to Labeled Attachment Score (LAS), Unlabeled SRL F1 (**U-SRL**) and Labeled SRL F1 (**L-SRL**). In this case, the dependency metrics are defined for a transformation of the trees that is possible because of the head information in each constituent. Notice, however, that not all metrics are applicable directly to all parsers.

First of all, the constituency metrics only apply to our parsers, as we are using a rather different grammar formalism. Our constituents are binarized and are labeled with the rule that was applied to form them. This is not the case for the only other Spanish constituency parser we could use, which belongs to the FreeLing suite. In this case, the constituents are not binarized and are annotated with a wide range of categories. The constituents in these trees tend to be very flat, not unlike AnCora. Given the complexity that a conversion between our format and FreeLing's constituency format would have, we decided to try a simpler approach that is doing the comparison only for dependencies, as FreeLing allows dependency outputs for all its parsers.

The dependencies comparison is not exempt from problems either, as different dependencies formalisms might differ in what elements they consider should be the heads of structures. For example, when transforming a prepositional phrase to dependencies, in our case we consider that the preposition is the head of the structure. The parsers in the FreeLing suite work in the same way. However, parsers based on Universal Dependencies like spaCy or UDPipe give precedence to the content words over the function words to assign them as heads, so in those cases the preposition would be a dependent instead of a head. In order to compare to these parsers, we post-processed the results and transformed the heads of prepositional phrases, copulas and other structures in order to adapt it to our format.

On the other hand, the labels we use in our dependency conversion are the grammar rules used to attach the dependant at each step, so they are a completely different label set from the ones used by the other parsers. The different parsers also use different dependency tag sets, even the parsers in the FreeLing suite have a different tagset (Treeler and LSTM use one tagset but Txala uses a different one), while spaCy and UDPipe use the same tagset defined by Universal Dependencies. Given this disparity of scenarios, we decided to only perform the full comparison with all the metrics for our parsing strategies, but only use the U-Dep metric when comparing to external parsers. The U-Dep metric is the most similar to standard Unlabeled Attachment Score, with the caveat that we are transforming the output of some parsers to comply to our definition of dependencies.

Lastly, the semantic role labeling metrics should be comparable across parsers because we defined them over them as dependencies between the heads of the constituents. Unfortunately, not all the parsers provide the SRL information, and in this case we can only compare to the parsers in the FreeLing suite.

[5] `http://treeler.lsi.upc.edu/`
[6] `https://spacy.io`

6.3 Syntactic parsing

The performance results over the test corpus for the different systems are shown in table 4. The best performing model according to this table is the LSTM top-down approach, both for constituency and dependency metrics. It is rather surprising that the CKY method underperformed for the constituency metrics, even when starting with the gold supertags which is the upper bound for the CKY methods. We can see that the unlabeled constituency metric gets about 77% for this method, while the corresponding dependency metric jumps to 92%. This might indicate that the main issue is that the method is able to identify the heads and dependents correctly, but misses the order of application of the rules frequently. The rather weak statistical model, than only takes in consideration partial information from the supertags, might be one reason why these models are underperforming. In the future we might try to enrich the statistical model using more fine-grained information like the lexical entries being used or the partial derivation history when applying the rules.

The LSTM top-down approach is very robust, outperforming all our other strategies and even getting results for dependency that are almost as good as with the CKY using gold tags. As expected, the bottom-up processes and the CKY using the supertagger results have much lower performance for all the metrics.

Compared to the external baselines, the results seem to indicate that our top-down parser also outperforms them for this corpus. However, we must take in consideration that the post-processing probably added some noise over the comparison, because the structures are not exactly the same as the original and also because we cannot be sure that the conversion of all structures that behave differently was done exhaustively.

Model	U-Cons	L-Cons	U-Dep	L-Dep
Bottom-up Bilex Context 0	49.39	38.51	63.49	60.10
Bottom-up Bilex Context 1	60.97	47.45	65.43	60.05
Bottom-up MLP Context 1	70.83	61.52	79.83	75.12
Bottom-up MLP Context 2	74.38	65.42	81.59	76.77
CKY Gold tags	*77.16*	*72.72*	*92.07*	*92.05*
CKY Supertagger	66.08	59.33	83.34	81.03
LSTM Top-down	**87.57**	**82.06**	**91.32**	**88.96**
FreeLing LSTM	-	-	83.15	-
FreeLing Treeler	-	-	83.61	-
FreeLing Txala	-	-	69.75	-
spaCy es_sm	-	-	83.01	-
spaCy es_md	-	-	83.69	-
UDPipe	-	-	82.09	-

Table 4: Results of the syntactic parsing experiments over the test set.

6.4 Semantic Role Labeling

The results for Semantic Role Labeling are shown in table 5. The upper bound in this case is given by the CKY using gold tags, getting around 88% for the SRL F1 metric, and a slight difference of one point between the unlabeled and labeled versions. It is expected that the unlabeled and labeled metrics would behave similarly for this case, because we are sure the parser starts with the appropriate supertags, so if it can predict the head-dependent pair correctly, it will certainly predict the appropriate semantic role for it as it is encoded in the supertag. The CKY using supertags, however, performs worse for detecting the head-dependent pair, and even worse for assigning the appropriate semantic role. This might be explained in part by the fact we observed empirically that the supertagger tends to predicts supertags without semantic role features more likely than the versions with semantic roles.

The LSTM top-down approach gets almost as good results for the unlabeled metric as the CKY with gold tags. However, it gets behind for the labeled metric, which indicates is frequently predicts a syntactic argument as semantic argument, but fails to assign the appropriate label.

Both CKY and top-down approaches outperform all the external baselines based on FreeLing, which behave more or less as our bottom-up baselines.

6.5 Execution time

Besides the global performance metrics, we compared the different parsers in terms of their speed based on the global execution time over our test corpus. This comparison is more related to the different type of algorithms used by each parser and might be skewed by how much fine-tuned or optimized they are. Table

Model	U-SRL	L-SRL
Bottom-up Bilex Context 0	59.73	45.82
Bottom-up Bilex Context 1	60.46	45.63
Bottom-up MLP Context 1	66.61	49.90
Bottom-up MLP Context 2	68.21	50.71
CKY Gold tags	*88.51*	*87.51*
CKY Supertagger	81.48	75.78
LSTM Top-down	**87.68**	**80.66**
FreeLing LSTM	68.50	60.74
FreeLing Treeler	69.10	61.53
FreeLing Txala	52.17	45.73

Table 5: Results of the semantic role labeling experiments over the test set.

6 shows the average time for parsing a sentence in the test set for the different models. The experiments were run on an Intel i7, 2.7GHz, 16GB RAM, without GPU acceleration. The metrics in the table are an average over all sentence lengths, with 1,692 sentences in total.

Model	Time (ms)
spaCy es_sm *(no SRL)*	9.3
Bottom-up Bilex Context 0	18.8
spaCy es_md *(no SRL)*	19.8
UDPipe *(no SRL)*	21.8
FreeLing Txala	41.8
FreeLing LSTM	60.3
Bottom-up MLP Context 1	63.9
Bottom-up MLP Context 2	78.0
LSTM Top-down	86.1
Bottom-up Bilex Context 1	88.3
CKY Gold tags	288.7
FreeLing Treeler	948.5
CKY Supertagger	1,237.3

Table 6: Average time in milliseconds for parsing a sentence in the test set.

There seems to be an advantage in terms of speed for the parsers based on neural networks over the parsers based on other (statistical) techniques, except our simplest bottom-up baseline, which is very fast (but, as we have seen, performs very poorly for the global metrics). The fastest parsers are spaCy and UDPipe (besides the simplest baseline), but we have to consider that these parsers only perform the syntactic dependency analysis, and they do not provide SRL information. This could be one of the reasons they are so fast. Around the middle of the table (between 40 ms and 90 ms per sentence) we can see the most of the parsers, including the LSTM top-down parser which is only slightly faster than the slower baseline. On the trailing positions we find the CKY parsers and FreeLing Treeler. The CKY with supertagger parser is the worst of all, taking more than a second for each sentence on average, which can be explained by the high number of times the back-off rules have to be enabled due to invalid combinations of supertags returned by the supertagger (around one word out of ten will have an incorrect supertag).

If we break down the execution time of the LSTM top-down process, we get that there is a balance in the time spent at each step: 54.1 ms for syntactic parsing and 31.9 ms for argument identification. This could be sped up if we used a unified architecture that could handle the splitting, rule prediction and semantic arguments identification steps at the same time. However, as seen in section 5.3, this architecture underperformed for other metrics, so we kept two separate networks.

The times shown in table 6 are average over all sentences in the test corpus, but given the differences between parsing algorithms, it would be interesting to see how well they behave for different sentence lengths. Figure 10 shows a breakdown of execution times for our approaches when parsing sentences of different length, up to 80 words long. In this case we show only the Bottom-up MLP2 baseline because it is the best performing of the simple baselines.

The LSTM top-down and the MLP Bottom-up execution times seem to grow close to linearly while for the CKY approaches it grows faster. One explanation for this might be that the growth rate for both LSTM top-down and MLP bottom-up (quasi-linear or quadratic) is lower than the one for CKY (at least cubic).

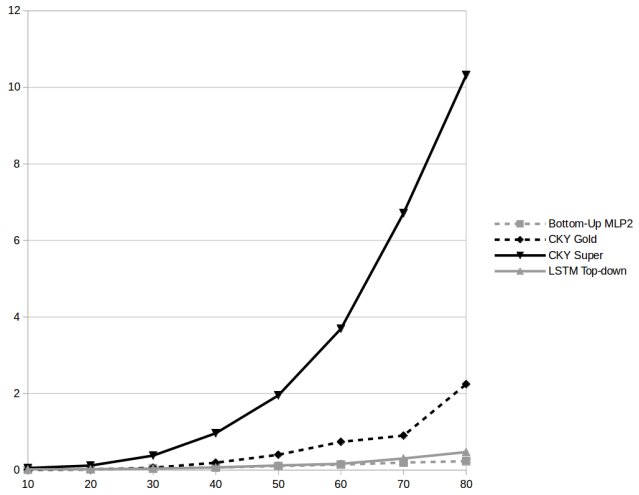

Figure 10: Breakdown of execution time in seconds for different input sizes.

Particularly the CKY with supertagger grows much faster than the others, which can be explained because, especially for longer sentences, the probability of obtaining a sequence of tags that does not form a correct tree is much higher as the sentence grows, so the fallback rules have to be enabled more frequently, rendering the process much slower.

6.6 Particular phenomena

		Bottom-up MLP 2	LSTM Top-down	CKY Gold	CKY Supert.	Freeling Txala	Freeling LSTM	Freeling Treeler	spaCy es_sm	spaCy es_md	UDPipe
Postponed Subjects	P	68.69	82.22	99.25	**84.25**	60.00	69.74	74.07	60.11	59.92	57.37
	R	31.18	64.02	98.89	**76.01**	19.92	65.49	62.73	56.45	59.59	52.39
	F	42.89	71.99	99.07	**79.92**	29.91	67.55	67.93	58.23	59.75	54.77
Clitics Identification	P	94.23	**98.76**	100.	98.71	96.44	78.25	80.67	84.99	83.47	82.33
	R	92.15	**99.03**	100.	95.46	85.83	88.58	88.44	97.38	97.93	96.83
	F	93.18	**98.90**	100.	97.06	90.82	83.09	84.38	90.76	90.12	89.00
Clitics Classification	Acc	66.16	**85.28**	95.46	83.63	75.10	80.47	80.88	-	-	-
	MP	17.95	**76.20**	98.46	65.05	72.03	82.67	83.50	-	-	-
	MR	23.91	**66.52**	88.18	57.80	44.02	51.51	54.60	-	-	-
	MF	20.50	**70.60**	92.78	61.00	49.20	56.37	58.63	-	-	-
Clitics Reduplication	P	0.00	32.69	100.	**75.00**	8.33	22.05	30.23	-	-	-
	R	0.00	**35.41**	81.25	18.75	2.08	31.25	27.08	-	-	-
	F	0.00	**34.00**	89.65	30.00	3.33	25.86	28.57	-	-	-
Relatives Identification	P	48.11	90.60	100.	**92.27**	72.84	76.59	78.23	69.16	70.85	66.49
	R	45.04	**89.00**	99.72	81.00	54.95	58.61	59.02	55.08	55.35	52.23
	F	46.53	**89.80**	99.86	86.27	62.64	66.41	67.28	61.32	62.05	58.51
Relatives Classification	Acc	16.69	**76.12**	81.14	57.67	4.21	43.15	42.06	-	-	-
	MP	10.72	**64.73**	84.36	54.32	30.54	40.75	40.76	-	-	-
	MR	8.17	**55.32**	78.47	37.82	7.63	24.78	24.39	-	-	-
	MF	8.34	**59.02**	72.39	43.29	6.54	29.18	28.62	-	-	-
Relative Referents	P	38.55	**73.61**	77.68	67.69	37.23	60.46	63.30	54.17	55.53	49.56
	R	36.09	**72.32**	77.47	59.43	28.08	46.26	47.76	43.14	43.55	38.94
	F	37.28	**72.96**	77.58	63.29	32.01	52.42	54.44	48.03	48.82	43.61
Coordinations Identification	P	30.57	**65.49**	74.02	54.92	24.48	56.53	56.09	41.39	42.47	37.22
	R	24.79	**65.22**	77.23	48.25	22.20	53.49	53.00	43.85	44.34	39.59
	F	27.38	**65.36**	75.59	51.37	23.28	54.96	54.50	42.59	43.38	38.37
Verb Phrases Identification	P	86.52	**92.47**	96.92	82.75	-	-	-	-	-	-
	R	66.10	**93.70**	87.77	69.73	-	-	-	-	-	-
	F	74.94	**93.08**	92.12	75.68	-	-	-	-	-	-
Impersonal Verbs Identification	P	51.95	**81.72**	91.05	68.80	-	-	-	-	-	-
	R	94.76	**95.33**	99.71	93.81	-	-	-	-	-	-
	F	67.11	**88.01**	95.18	79.38	-	-	-	-	-	-

Table 7: Results of the experiments for some language phenomena in Spanish. We show precision, recall and F1 score for identification tasks, and accuracy and macro metrics for classification tasks.

We described in section 3 the grammar we are using and mentioned some of the Spanish phenomena we are trying to model, which in some cases led us to make particular choices for designing the rules and features. We now want to analyze to what extent our parsers are able to capture these phenomena in actual sentences. In this section we will define a set of metrics over some particular Spanish phenomena, which cover some aspects of the language we wanted to address. As was the case with the dependency results

conversion, in these cases it was also necessary to post-process the results of the external parsers so as to adapt the different ways they represent these phenomena. Once again, this post-processing might add some noise in the results, so the comparisons shown in this section might hint at some differences between parsers, but further research would be needed to confirm these results.

- Postponed subjects: Identifying a subject that occurs on the right of the verb, as opposed to the more usual left position.
- Clitics identification and classification: Detecting the clitic pronouns that accompany a verb and classifying them according to SRL.
- Clitics reduplication: In Spanish, clitic pronouns can be used in lieu of an explicit object, but it is also possible to include both the object and the corresponding clitic at the same time. This is called clitic reduplication or clitic doubling.
- Relatives identification and classification: Detecting the relative pronouns and expressions that are attached to some verbs creating relative sentences, and classifying them according to SRL.
- Relatives referent identification: Besides identifying the relative pronoun and its verb, the parser must also properly identify the nominal referent for the relative pronoun.
- Coordinations identification: Finding chains of (two or more) coordinated elements.
- Verb Phrases Identification: Finding chains of verbs that form a compound, such as *"había ido"* (*"had gone"*) or *"comenzó a cantar"* (*"started to sing"*).
- Impersonal Verbs Identification: Finding verbs that should not be attached to a subject such as *"hay"* (*"there is"*) or *"llueve"* (*"it rains"*). Notice that in English it is mandatory that these verbs have a subject nonetheless.

Table 7 shows the results of the different parsers for these particular phenomena. Notice that in most cases the LSTM top-down strategy has the best results, except for the postponed subject detection, where CKY with supertags behaves better.

7 Conclusions

We set out this research with the objective of creating a statistical HPSG parser for Spanish. This was accomplished by completing a series of steps:

We designed a HPSG grammar adapted to Spanish. This grammar is different than the previously existing Spanish HPSG grammar (SRG [9]) in that our grammar is built with the explicit design objective of creating a statistical parser. Because of this, the rules in our grammar tend to be very broad and capture a great number of situations, and we leave the finer-grained task of distinguishing between good uses and bad uses of the language to the statistical modules. In particular, we needed to be able to correctly model the situations present in the corpus we would use (Spanish AnCora). Compared to the original standard HPSG grammar, ours adds some features for Spanish like the use of clitic pronouns, and some particularities for modeling verb phrases.

We transformed the AnCora corpus of Spanish sentences into our HPSG format. We first used an automated process that transversed the corpus simplifying complex structures and using heuristics to detect heads and rules to apply for each constituent, then we analyzed the resulting sentences using our feature structure implementation of the grammar, creating a proper HPSG version of the corpus. The final corpus contains 517,237 instances of words (tokens) in 17,348 sentences. These words correspond to 83,594 unique lexical entries, which are structured in 3,245 lexical frames. The corpus was split in standard training, development and test partitions of around 80%, 10% and 10%.

We implemented several parsing algorithms for our grammar trained over the data from our corpus. The algorithms belong to these three categories: A bottom-up strategy that compares consecutive pairs of words in a sentence until finding the pair that is most likely to form a constituent, and repeats the process recursively until forming the whole tree; a statistical CKY approach that uses the lexical entries encoded as supertags and a PCFG style statistical model for guiding the CKY algorithm; and finally a top-down approach that recursively analyzes a sequence of words using a LSTM neural network and splits the sequence in the most likely point to form appropriate constituents and then uses a second neural network that predicts the semantic arguments.

We evaluated the performance of our parsing strategies and compared them to some well established Spanish parsers over the test corpus. We found that the LSTM top-down strategy outperforms all the other

baselines in terms of constituency (87.57 U-Cons, 82.06 L-Cons), dependency (91.32 U-Dep, 88.96 L-Dep) and semantic role labeling metrics (87.68 U-SRL, 80.66 L-SRL). However, we must take in consideration that the results of the external parser had to be post-processed to comply to our format, and this could have added some noise to the comparison. In terms of speed, the LSTM top-down approach is faster than than the CKY parser, but lags behind other parsers such as spaCy or UDPipe. We also evaluated the parsers using a set of metrics designed to test their performance on some particular Spanish phenomena. The LSTM top-down parser was the best for most of these metrics: detecting (98.90 F1) and classifying (70.60 macro-F1) clitics, and also detecting cases of clitics reduplication (34.00 F1), detecting (89.60 F1) and classifying (59.02 macro-F1) relative clauses that modify a noun phrase, identifying the referents of these constructions (72.96 F1), detecting chains of coordinations (65.36 F1), identifying verbal periphrases (93.08 F1) and detecting impersonal verbs (88.01 F1); while CKY with supertags was the best parser for detecting postponed subjects (79.92F1).

During the course of the project we found several directions in which we think this work could be improved, that were not explored due to time limitations. For example, there are many possible improvements in the parsing process, such as unifying the two-step process of top-down parsing followed by SRL classification into a single neural network architecture in order to make it faster, or combining the LSTM top-down with the supertagger approach in order to leverage the fine-grained subcategorization information present in the supertags to aid the parsing process as a whole.

As mentioned before, our grammar only models one type of long range dependency, which is the relative clause that modifies a noun phrase, but there is information in AnCora that might allow us to incorporate knowledge on longer distance dependencies. This would imply changes in the grammar adding a filler-gap mechanism that could leave any number of gaps in some parts of the structure that are filled with elements found in other parts of the tree, so further research is needed to understand how this could be integrated to a top-down parsing strategy, or if a combination of top-down and bottom-up heuristics might be needed.

We would also like to find ways of combining our grammar and parsers with the rich linguistic information used in SRG [9], and also try to find ways to port our strategies to other languages. For example, it could be possible to combine constituency and dependency corpora for the same language (similar to the idea in [15]) to create a basic HPSG format that could be used as starting point for our parsing process.

References

[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc., 2009.

[2] L. Chiruzzo, "Statistical Deep Parsing for Spanish," Ph.D. dissertation, Facultad de Ingeniería Universidad de la República, Uruguay, 2020.

[3] C. Pollard and I. A. Sag, *Head-driven Phrase Structure Grammar*. University of Chicago Press, 1994.

[4] M. Steedman, "A very short introduction to CCG," *Unpublished paper. http://www.coqsci.ed.ac.uk/steedman/paper.html*, 1996.

[5] A. K. Joshi, "Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?" *Cambridge University Press*, 1985.

[6] M. Dalrymple, *Lexical functional grammar*. Brill, 2001.

[7] N. Chomsky, "Bare phrase structure," *Evolution and revolution in linguistic theory*, pp. 51–109, 1995.

[8] M. Marimon, "The Spanish Resource Grammar." in *Proceedings of the International Conference on Language Resources and Evaluation, LREC*, Valletta, Malta, 2010, pp. 17–23.

[9] M. Marimon, N. Bel, S. Espeja, and N. Seghezzi, "The Spanish Resource Grammar: pre-processing strategy and lexical acquisition," in *Proceedings of the Workshop on Deep Linguistic Processing*. Association for Computational Linguistics, 2007, pp. 105–111.

[10] M. Marimon, N. Bel, and L. Padró, "Automatic selection of HPSG-parsed sentences for Treebank construction," *Computational Linguistics*, vol. 40, no. 3, pp. 523–531, 2014.

[11] T. Ninomiya, T. Matsuzaki, Y. Tsuruoka, Y. Miyao, and J. Tsujii, "Extremely lexicalized models for accurate and fast HPSG parsing," in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2006, pp. 155–163.

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[13] Y. Miyao, T. Ninomiya, and J. Tsujii, "Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank," in *Natural Language Processing–IJCNLP 2004*. Springer, 2005, pp. 684–693.

[14] C. Bonial, O. Babko-Malaya, J. D. Choi, J. Hwang, and M. Palmer, "PropBank annotation guidelines," *Center for Computational Language and Education Research Institute of Cognitive Science University of Colorado at Boulder*, 2010.

[15] J. Zhou and H. Zhao, "Head-driven Phrase Structure Grammar Parsing on Penn Treebank," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2396–2408. [Online]. Available: https://www.aclweb.org/anthology/P19-1230

[16] M. Taulé, M. A. Martí, and M. Recasens, "AnCora: Multilevel Annotated Corpora for Catalan and Spanish." in *Lrec*, 2008.

[17] L. Chiruzzo and D. Wonsever, "Spanish HPSG Treebank based on the AnCora Corpus," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, 2018.

[18] I. A. Sag, T. Wasow, and E. M. Bender, *Syntactic theory: A formal introduction*, 2nd ed. Center for the Study of Language and Information Stanford, CA, 2003.

[19] L. Pineda and I. Meza, "The Spanish pronominal clitic system," *Procesamiento del lenguaje natural*, vol. 34, pp. 67–103, 2005.

[20] L. Chiruzzo, "Construcción de Recursos Lingüísticos para una Gramática HPSG para el Español," Tesis de maestría, Universidad de la República, Uruguay, 2015.

[21] J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štepánek, P. Straňák, M. Surdeanu, N. Xue, and Y. Zhang, "The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*. Association for Computational Linguistics, 2009, pp. 1–18. [Online]. Available: http://aclweb.org/anthology/W09-1201

[22] T. Matsuzaki, Y. Miyao, and J. Tsujii, "Efficient HPSG Parsing with Supertagging and CFG-Filtering." in *IJCAI*, 2007, pp. 1671–1676.

[23] T. Kasami, "An efficient recognition and syntax-analysis algorithm for context-free languages," *Coordinated Science Laboratory Report no. R-257*, 1966.

[24] D. H. Younger, "Recognition and parsing of context-free languages in time n3," *Information and control*, vol. 10, no. 2, pp. 189–208, 1967.

[25] J. Cocke, *Programming languages and their compilers: Preliminary notes*. New York University, 1969.

[26] A. K. Joshi and B. Srinivas, "Disambiguation of super parts of speech (or supertags): Almost parsing," in *Proceedings of the 15th conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1994, pp. 154–160.

[27] J. R. Curran, S. Clark, and D. Vadas, "Multi-tagging for lexicalized-grammar parsing," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 697–704.

[28] R. Dridan, "Using lexical statistics to improve hpsg parsing," Ph.D. dissertation, University of Saarland, 2009.

[29] L. Chiruzzo and D. Wonsever, "Building a supertagger for Spanish HPSG," *Computer Speech & Language*, vol. 54, pp. 44–60, 2019.

[30] ——, "Syntactic Analysis and Semantic Role Labeling for Spanish using Neural Networks," in *CICLing 2019 - 20th International Conference on Computational Linguistics and Intelligent Text Processing*, 2019.

[31] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[33] L. Chiruzzo and D. Wonsever, "Statistical Deep Parsing for Spanish Using Neural Networks," in *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*. Online: Association for Computational Linguistics, July 2020, pp. 132–144. [Online]. Available: https://www.aclweb.org/anthology/2020.iwpt-1.14

[34] L. Padró and E. Stanilovsky, "Freeling 3.0: Towards wider multilinguality," in *LREC2012*, 2012.

[35] J. A. Batalla, E. C. Pujadas, and A. Mayor, "TXALA un analizador libre de dependencias para el castellano," *Procesamiento del Lenguaje Natural*, vol. 35, 2005.

[36] M. Lloberes, I. Castellón, and L. Padró, "Spanish FreeLing Dependency Grammar." in *LREC*, vol. 10, 2010, pp. 693–699.

[37] M. Straka and J. Straková, "Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe," in *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, August 2017, pp. 88–99. [Online]. Available: http://www.aclweb.org/anthology/K/K17/K17-3009.pdf