A Truly Joint Neural Architecture for Segmentation and Parsing

Danit Yshaayahu Levi **Reut Tsarfaty**
Bar Ilan University, Ramat Gan 5290002, Israel
danity251@gmail.com reut.tsarfaty@biu.ac.il

Abstract

Contemporary multilingual dependency parsers can parse a diverse set of languages, but for Morphologically Rich Languages (MRLs), performance is attested to be lower than other languages. The key challenge is that, due to high morphological complexity and ambiguity of the space-delimited input tokens, the linguistic units that act as nodes in the tree are not known in advance. Pre-neural dependency parsers for MRLs subscribed to the *joint morpho-syntactic hypothesis*, stating that morphological segmentation and syntactic parsing should be solved jointly, rather than as a pipeline where segmentation precedes parsing. However, neural state-of-the-art parsers to date use a strict pipeline. In this paper we introduce a joint neural architecture where a lattice-based representation preserving all morphological ambiguity of the input is provided to an arc-factored model, which then solves the morphological segmentation and syntactic parsing tasks at once. Our experiments on Hebrew, a rich and highly ambiguous MRL, demonstrate state-of-the-art performance on parsing, tagging and segmentation of the Hebrew section of UD, using a single model. This proposed architecture is LLM-based and language agnostic, providing a solid foundation for MRLs to obtain further performance improvements and bridge the gap with other languages.

1 Introduction

Dependency parsing is the task of automatically analyzing the syntactic structure of a sentence and exposing the functional relationships between its words. In the past, dependency parsing was shown to be extremely useful for many language processing tasks, as machine translation (Galley and Manning, 2009), question answering (Garimella et al., 2021) and information extraction (Hwang et al., 2020), to name a few. While nowadays many English NLP tasks are solved end-to-end using large language models (LLMs) and without accessing any symbolic structure, for low- and medium-resource languages, parsers are still indispensable, enabling a host of downstream applications.

Most neural state-of-the-art dependency parsers to date presuppose a pipeline architecture (Qi et al., 2020; Honnibal and Montani, 2017; Minh Nguyen and Nguyen, 2021) that includes several analysis stages — tokenization, word segmentation, part-of-speech (POS) tagging, morphological feature tagging, dependency parsing, and sometimes also named entity recognition — and the linguistic features from each stage are provided as input to the tasks that follow it, and contribute to the overall efficacy.

In morphologically-rich languages (MRLs), many raw space-delimited tokens consist of multiple units, each of which serves a distinct role in the overall syntactic representation (Tsarfaty et al., 2010, 2020). Consequently, segmentation is essential for accurate MRL parsing. However, due to high morphological ambiguity, when segmentation is performed prior to (and independently of) the parsing phase, segmentation errors may propagate to undermine the syntactic predictions, and subsequently lead to an overall incorrect parse.

According to the joint hypothesis, that was heavily populated in parsing studies for MRLs in the pre-neural era (Tsarfaty, 2006; Cohen and Smith, 2007; Goldberg and Tsarfaty, 2008; Green and Manning, 2010; Seeker and Çetinoğlu, 2015), morphological segmentation and syntactic predictions are mutually dependent, and hence, these two tasks should be solved together.

Following these lines, More et al. (2019) developed a joint morpho-syntactic transition-based parser that achieved state-of-the-art (SOTA) results on Hebrew parsing. This system employs a morphological lattice as input for a transition system, with both syntactic and morphological transitions, for picking the right arcs and segments in tan-

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 1408–1420

dem. Another influential work is that of Seeker and Çetinoğlu (2015), that displays all potential segments that could be involved in any token's analysis, and allows an MST graph-based parser pick the highest scoring subset of arcs and nodes as the output dependency tree, showing parsing improvements for both Hebrew and Turkish. Similarly, Krishna et al. (2020a,b) contributed a non-neural graph-based parser for Sanskrit which employ energy-based modeling to determine the optimal path on a graph which jointly represents a valid segmentation and syntactic analysis. However, this architecture is non-neural and Sanskrit specific, relying on labor-intensive hand-crafted feature engineering. In neural settings, and still for Sanskrit, Sandhan et al. (2021) introduced a pre-training approach which focuses on amalgamating word representations generated by encoders trained on auxiliary tasks, such as morphological and syntactic tags. Unlike the present work, this neural architecture does not make any segmentation decisions nor does it leverage the lattice structure for joint segmentation and parsing.

All in all, in the case of neural multilingual dependency parsers, the pipeline approach of segment-then-parse is fully maintained (Kondratyuk and Straka, 2019a,b; Minh Nguyen and Nguyen, 2021), and no language-agnostic architecture for morphological segmentation and syntactic parsing is yet to be found.

In this paper we revisit the *joint hypothesis* as a viable way to improve neural dependency parsing for MRLs. The idea, in a nutshell, is as follows. We start off with an arc-factored model (Dozat and Manning, 2017) that accepts a sequence of words as input, and generates a dependency tree by picking the highest scoring arcs connecting all words. In our approach, the arc-factored model takes as input a linearized lattice containing all possible morphological segments that may *potentially* act as nodes, and learns to assign a head and label to each such node. During training, incorrect segments are mapped to an auxiliary node, of which subtree is excluded from the final dependency tree. At inference time then, the model maps relevant segments to the main branch and unused segments to the auxiliary branch, building a complete dependency tree. In this process, morphological segmentation decisions get informed by the syntactic arcs, and vice versa. We further extend the architecture with a multi-task learning (MTL) component to predict the features of each node, e.g., *POS, gender, number* and *person*.

Our experiments on the Hebrew Section of UD[1] show that in cases where the input morphological analyses are complete, our model provides new and improved state of the art results for segmentation and parsing for Hebrew, in a single, jointly trained, model. In the more realistic case, where some of the word lattices may lack possible analyses (the case of out of vocabulary (OOV) tokens), the model still delivers competitive results for segmentation, tagging and parsing, outperforming the state-of-the-art results of the de-facto standard pipelines, Stanza and Trankit.

2 Challenge and Research Objectives

The goal of dependency parsing is to automatically analyze the syntactic structure of a sentence by indicating the functional relationship between its words. The input is assumed to be a sequence of space-delimited tokens that represent words, and the output is a directed tree where each input word serves as a node, and each arc represents a relation between two such words. An arc can be labeled to indicate the relation type between the two words.

Deep neural networks have recently achieved unprecedented results in many areas of natural language processing, including the dependency parsing task. The architecture of Dozat and Manning (2017) (that followed up on Kiperwasser and Goldberg (2016)) is currently accepted as the standard architecture for dependency parsing. Dozat and Manning present a simple neural architecture where an arc factored model selects the best set of dependency arcs and labels. This approach is the foundation of several SOTA dependency parsers, including Stanza (Qi et al., 2020) and Trankit (Minh Nguyen and Nguyen, 2021), which have been trained and successfully used across different languages. Crucially, these parsers and others (Dozat and Manning, 2017; Qi et al., 2020; Minh Nguyen and Nguyen, 2021; Kondratyuk and Straka, 2019a,b) all subscribe to a pipeline approach, where the input tokens are pre-segmented, and these segments uniquely determine the nodes in the tree.

This pipeline approach has been applied across many language types, including morphologically rich languages (MRL). However, MRLs pose a significant challenge to such architectures. In a

[1]The UD initiative https://universaldependencies.org/treebanks/he_htb/index.html

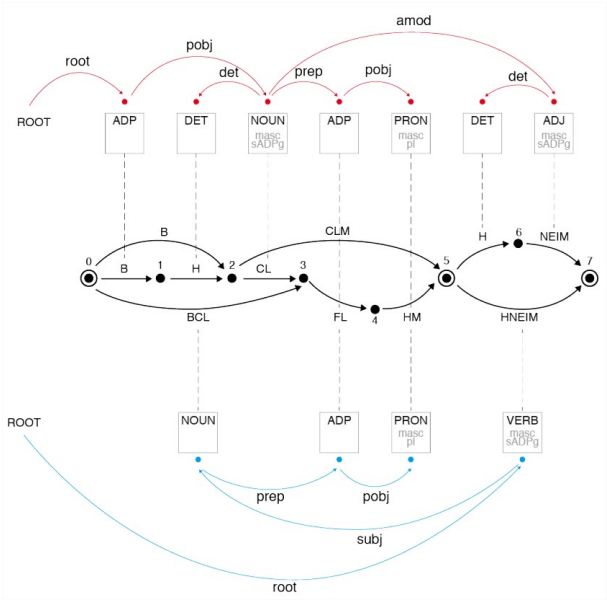

Figure 1: The morphological lattice for the Hebrew phrase *bclm hneim* and two associated dependency trees depicting alternative segmentations (Origin: More et al. (2019)). The upper tree illustrates the syntactic structure corresponding to "In their pleasant shadow", while the lower tree corresponds to "Their onion was pleasant". This highlights the existence of multiple morphological decompositions and various potential dependency trees.

pipeline architecture, where morphological segmentation is performed prior to parsing. However, tokens in MRLs are rich and complex, and include multiple units that can act as individual nodes in the tree. Hence their segmentation may be highly ambiguous, and the nodes of the tree are not known in advance. When these segments are fixed prior to parsing, wrong segmentation seriously hinders parsing results. The main challenge is then to find the appropriate segmentation that is relevant to the particular syntactic context. This challenge is illustrated at Figure 1. here we consider the Hebrew phrase '*bclm hneim*', which can be translated in various ways depending on the segmentation analysis applied: "In their pleasant shadow", "In the pleasant photographer," or "Their onion was pleasant". Figure 1 provides a lattice representation of all morphological analyses of the phrase, where different segmentations give rise to substantially different syntactic trees.

Pre-neural models addressed this challenge by jointly modeling morphological segmentation and dependency parsing, and have shown that it yields superior results for both tasks. The pressing research question at hand is whether this hypothesis can also be validated within the context of neural parsing architectures. In other words, can neural parsing models benefit from a joint approach to segmentation and parsing, similar to what has been observed in non-neural models?

This paper addresses two primary objectives. Firstly, we aim to introduce a unified neural architecture that jointly solves segmentation, tagging and parsing, with the aim of empirically validating the joint hypothesis within the realm of neural architectures. Secondly, we seek to attain state-of-the-art (SOTA) results for Hebrew, a language renowned for its formidable parsing challenges attributed to its substantial morphological ambiguity.

3 The Proposal: A Model for Joint Morphological Segmentation and Syntactic Dependency Parsing

Task Definition Formally, our proposed model is defined as a structure prediction function $f : S \rightarrow D$, where $s \in S$ represents a sequence of raw input tokens, and $d \in D$ denotes a dependency tree with nodes corresponding to disambiguated units, which we refer to here as *morphological segments*. Crucially, we retain morphological ambiguity, and deliver all possible analyses of s to the parser. Hence, we assume a Morphological Analyzer (MA), that given an input sentence $s = s_1, \ldots, s_k$ yields a *token-lattice* termed $L_i = MA(s_i)$ for each token s_i. The complete lattice of the input sentence $L_s = MA(s)$ is defined as the concatenation of the token lattices $L_s = MA(s_1) \circ \cdots \circ MA(s_k)$. Our structure prediction function becomes $f : L \rightarrow D$, with $L_s \in L$ as the morphological lattice of $s \in S$.

Input Linearization Upon receiving an input lattice L_s, we aim to linearize it in order to be able to encode it as an input vector for the neural architecture. As shall be seen shortly, the linearization is a critical phase for obtaining a neural encoding of the non-linear, morphologically ambiguous, input. We illustrate the linearization process using the Hebrew sentence '*bkrti bbit hlbn*' (lit: "I-visited in-the-house the-white", trans: "I visited the white house"). Initially, the MA provides a list of all potential analyses for each token: bkrti: [('bkrti')], bbit: [('b', 'bit'), ('b', 'h', 'bit')], hlbn: [('h', 'lbn'), ('hlbn')]. Subsequently, each token is linearized independently: bkrti: ['bkrti'], bbit: ['b', 'bit', 'b', 'h', 'bit'], hlbn: ['h', 'lbn', 'hlbn']. Finally, all linearized analyses are concatenated according to the initial order: ['bkrti', 'b', 'bit', 'b', 'h', 'bit',

'h', 'lbn', 'hlbn'].

Formally, the *linearize* function takes an input lattice L_s and returns a sequence of m analyses while maintaining the partial order of the tokens. Within the input lattice $L_s = MA(s_1) \circ \cdots \circ MA(s_k)$, each $MA(s_j)$ encompasses a comprehensive set of potential analyses — segmentation options — for the token s_j. Let k_j be the number of analyses of the j^{th} token; then, it holds that $\sum_{j=1}^{n} k_j = m$. Also, let a_j^i be the i^{th} analysis of the j^{th} token. The linearization function works as follow:

$$linearize(L_t) = linearize(MA(t_1) \circ \cdots \circ MA(t_k)) = linearize(MA(t_1)) \circ \cdots \circ linearize(MA(t_k)) = a_1^1, \ldots, a_1^{k_1} \circ a_2^1, \ldots, a_2^{k_2} \circ \cdots \circ a_n^{k_n}$$

The number of morphemes in an analysis a_i^j is $r_{(i,j)}$, denoted as

$$a_i^j = m_{(i,j)}^1, \ldots, m_{(i,j)}^{r_{(i,j)}-1}, m_{(i,j)}^{r_{(i,j)}}$$

Consequently, the total number of morphemes in the linearized lattice is given by $\sum_{j=1}^{n} \sum_{i=1}^{k_j} r_{(i,j)}$. Thus, the linearized lattice can be expressed as a sequence of morphemes:

$$linearize(L_t) = m_{(1,1)}^1, \ldots, m_{(1,1)}^{r_{(1,1)}}, \ldots, m_{(n,k_n)}^{r_{(n,k_n)}}$$

Joint Prediction We extend the simple and well-known neural arc-factored model to accept a linearized Lattice as input, and choose a subset of arcs with the highest scores as the output dependency tree. Crucially, this selected set of arcs does not take all segments as nodes. On the contrary, the arc selection essentially determines which segments from the lattice are included in the final tree, and is subject to certain constraints, as we detail shortly.

Let us define $\mathcal{A}(L_s)$ the set of all possible subsets of arcs in the linearized lattice. In our model we aim to select a highest scoring subset A as the DEP tree:

$$DEP = argmax_{A \in \mathcal{A}(L_s)} score(A)$$

To ensure that the nodes of the selected arcs in A form a valid morpheme sequence, the nodes that participate in the subset of arcs must adhere to the following constraints:

1. Exactly one segmentation analysis should be chosen per token.
2. All morphemes in the chosen analysis should be included in the selection.
3. Arcs cannot connect morphemes from *different* analyses of *the same* token.

When enforcing these constraints, the set of subsets $\mathcal{A}(L_s)$ is significantly smaller than a straightforward cartesian product over all possible segments. We thus define the set C:

$$C = constrained(\mathcal{A}(L_s)) = \{a_1^1, \ldots, a_1^{k_1}\} \times \{a_2^1, \ldots, a_2^{k_2}\} \times \cdots \times \{a_n^1, \ldots, a_n^{k_n}\}$$

And the prediction function becomes

$$DEP = argmax_{A \in C} score(A)$$

Finally, note that in this model, A is not formally defined to form a tree. In order to ensure a tree structure, at inference time we employ an Maximum Spanning Tree algorithm (MST) on top of the constrained graph.

$$DEP = argmax_{A \in C} MST\_score(A)$$

Note that the highest scoring tree uniquely defines the set of nodes that participate in it, so the MST in this proposed method also acts to substantiate the scoring function for morphological disambiguation (MD). We thus get:

$$\langle MD, DEP \rangle = argmax_{A \in C} MST\_score(A)$$

4 The Overall Architecture

The Joint Arc-Factored Model The central component of the architecture is an arc-factored model capable of selecting the highest-scoring subset from the (constrained) set of arcs. Our departure point is the Biaffine-Score architecture of Dozat and Manning (2017) , which is in turn based on Kiperwasser and Goldberg (2016), and is currently the de factor standard architecture for dependency parsing. In order to turn this architecture into a joint segmentation-parsing prediction model, we introduce several novelties into Dozat and Manning (2017).

In the original architecture, the input consists of a tokenized sentence with an additional root token. However, for joint prediction, we modify the input to be the linearized lattice of the input sentence

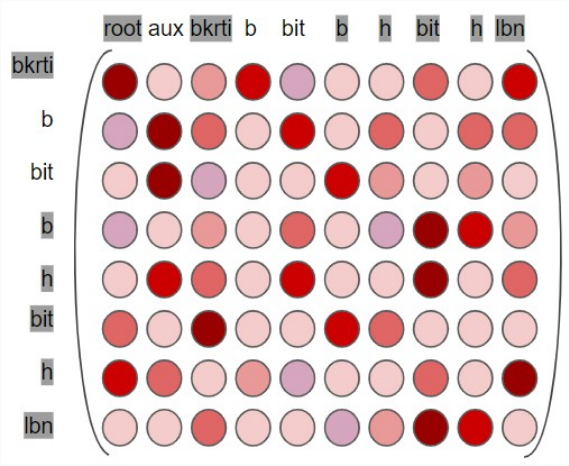

Figure 2: The Head matrix of the Hebrew sentence *bkrti bbit hlbn*. Each row depicts the scores assigned to all heads of a particular segment (including the root and auxiliary tokens). The darker color indicates a higher score. The input to Dozat and Manning's original architecture consists of the root and gray-marked segments.

and add two nodes: the *root node* and an *auxiliary node*. The linearized lattice represents the list of all segments from all possible analyses, ordered as detailed in Section 3.

During training, the head of any segment that does not appear in the gold dependency tree is attached under the *auxiliary* token, and thus, the model learns to assign to the *root* only the segments of the relevant morphological analyses in context. At inference times, the irrelevant segments are assigned to the auxiliary token, and the auxiliary subtree is removed altogether, leaving a single rooted dependency tree intact. Figure 2 describes the head matrix of the Hebrew sentence '*bkrti bbit hlbn*' in our model. The gray-marked segments participate in the final tree.

Input Embeddings The input to our proposed architecture consists of the contextualized embeddings of the segments sequence, including also the root and auxiliary tokens at the beginning of that sequence. Unlike the original input, the linearized lattice representation lacks a coherent context for generating high-quality contextualized embeddings. Therefore, we establish a valid context for each token analysis from the original sentence, and we employ contextualized embedding that reflect this context.

To create the embeddings for each of the input segments, we begin with the original sequence of tokens $s_1, s_2, \ldots, s_k$. For each analysis a^i_j, we create an analysis where we replace s_j with the sequence of morphological segments and results in the following sequence:

$$s_1, \ldots, s_{j-1}, m^1_{(i,j)}, \ldots, m^{r_{(i,j)}}_{(i,j)}, s_{j+1}, \ldots, s_k$$

Using this modified context, we obtain contextual embeddings for each of $m^1_{(i,j)}, m^2_{(i,j)}, \ldots, m^{r_{(i,j)}}_{(i,j)}$ using an LLM encoder. We apply the LLM's original tokenizer to the morpheme sequence. if the morpheme is present in the LLM's encoder vocabulary, it remains untokenized and receives a single vector embedding. Conversely, for out-of-vocabulary morpheme, tokenization is carried out based on the LLM's encoder. Each new token receives a vector, and the vector of the first token represents the whole original morpheme.

The embeddings of the root and auxiliary tokens are directly derived from the original token sequence.[2]

The Arc Selection Phase The encoded segments are inputted into Dozat and Manning's architecture, which produces two matrices: one for head prediction and another for label prediction. The head prediction matrix assigns a probability to each pair of segments, including the root and auxiliary tokens, indicating whether one is the head of the other. A similar process is carried out for each pair of tokens with respect to every possible label. The architecture by Dozat and Manning (2017) for dependency parsing remains unchanged, and the matrices are employed to represent the syntactic relationships between segments. The introduction of an auxiliary token allows for the exclusion of specific segments from the final tree by removing all segments for which it serves as the head. All other segments are retained as nodes in the final tree. This modification enables the architecture to perform joint segmentation and parsing predictions.

Input Constraints To ensure that the output conforms to the constraints outlined in section 3, it is imperative to limit the segment-sets that can form trees constructed beneath the root.

To implement the *constrained* function, we adopt a strategy where only one analysis per token is selected in each possible tree. In cases where the highest scoring subset selects more than one analysis per token, or if no segments from its analyses

[2]During the embedding process, we may generate different embedding vectors for segments with identical forms, e.g., the morph b repeats twice in the matrix in Figure 2. However, these identical forms reside in the context of different token analyses, and thus their contextualized embeddings are different.

were selected, we opt for the next best analysis, which contains the head segment with the highest score. When a particular analysis is chosen, we mask the auxiliary token for each of the segments, ensuring that all of them are included in the final tree. Finally, we mask all arcs where a segment in a chosen analysis that relates to a segment in a different analysis of the same token.

Multitask Learning We aimed to leverage additional linguistic tasks such as gender, person, number, and POS. Consequently, we expanded upon the original architecture introduced by Dozat and Manning to accommodate these MTL objectives. The input embedding is processed through a BiLSTM, and the output is utilized by both the aforementioned joint architecture and the MTL architecture designed to handle these linguistic tasks.

The Overall Architecture Figure 3 presents the proposed architecture from a bird's eye view. We illustrate it for the phrase '*bbit hlbn*' ("in the white house") from Figure 2 which presented the head matrix of the phrase.

The architecture of our model begins by taking a sentence and embedding its linearized lattice representation. These embeddings then undergo processing through a two-layer BiLSTM. The BiLSTM's output is further directed into two distinct BiLSTMs: one for the Biaffine score architecture and another for the MTL architecture. Within the Biaffine module, it is utilized to generate head and label matrices, facilitating the prediction of a well-structured dependency tree. In the MTL segment, the output undergoes dimension reduction through a linear layer. Subsequently, an additional and separate linear layer is applied for each MTL task (POS, gender, number, and person) to predict language features. We compute the loss using the *cross-entropy* function for the head, label, and each of the MTL tasks, and then aggregate them into a combined loss.

5 Experimental Setup

Goal We set out to evaluate the performance of the proposed joint architecture on segmentation, tagging and parsing. In all experiments, we show the segmentation (SEG) and dependency parsing (DEP) $F_1 score$. Additionally, for experiments with the POS MTL, we present the POS $F_1 score$.

Data All our experiments were trained and tested on the standard split of the Hebrew section of the

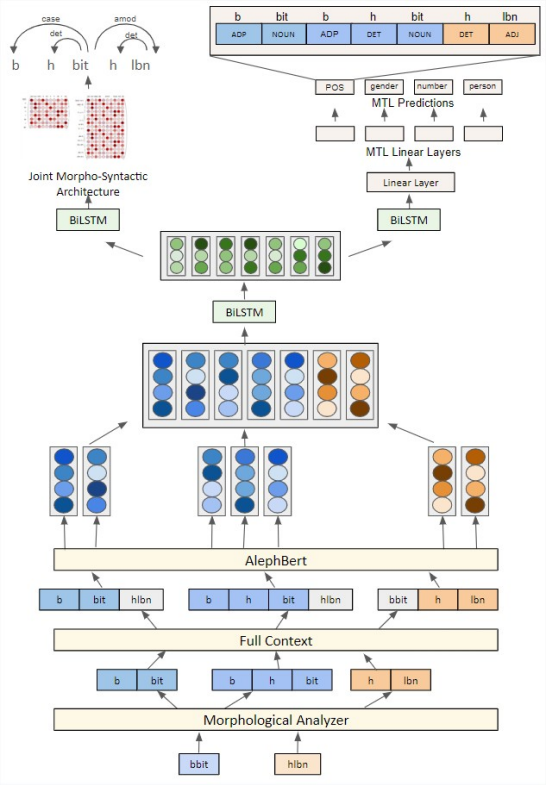

Figure 3: The comprehensive architecture examines the phrase '*bbit hlbn*' (in-the-house the-white), encompassing the processes of morphological analysis, generating context for each analysis, acquiring contextualized embeddings, constructing a dependency tree, and predicting linguistic features.

UD treebank collection (Nivre et al., 2016). The training set, dev set, and test set consist of 5,168 sentences, 484 sentences, and 491 sentences, respectively. The morphological analysis for the input of our model is provided by the Morphological Analyzer (MA) of More et al. (2019).[3] The MA provides the segmentation, Part-of-Speech (POS) tags and morphological features for each segment in each one of the possible analyses.[4]

Embeddings The way we generate embedding for the input lattice complements the architectural design and significantly impacts the parser performance. Alongside our proposed sentence-contextualized embedding (Section 4), we assessed two alternative techniques. We tested this structure with AlephBert's (Seker and Tsarfaty, 2021) static (Static) enbeddings and contextualized (Contextualized) embeddings generated directly for the

[3] https://github.com/OnlpLab/yap

[4] To address situations where a segment has multiple potential POS tags or morphological features, we employ a criterion based on the most common label (or the first one in case of a tie).

linearized lattices.

Evaluation Scenarios As part of our model we use a Morphological Analyzer (MA) component for generating the lattices. However, any realistic MA is not exhaustive, as it may lack some analyses, with certain tokens entirely missing (out of vocabulary, OOV). Hence, we aim to gauge the effect of the MA coverage on the parser performance. In the **Infused** scenario, we experiment in a setup where for all sentences the correct analysis is guaranteed to be incorporated as one of the lattice's internal paths. To establish the infused scenario, we examine all tokens in the dataset and integrate the gold analysis for each token back into the MA. In the **Uninfused** scenario, which represents a realistic scenario, we use the MA as is, and there may be missing analyses in the constructed lattices at inference time.

Models Current SOTA results in Hebrew dependency parsing are presented by Trankit (Minh Nguyen and Nguyen, 2021) and Stanza (Qi et al., 2020), both of which are multilingual neural parsers. Since our proposed architecture essentially extends that of Dozat and Manning (2017), we also evaluate their architecture in a pure pipeline setting. In this paper, we introduce three variations of Hebrew parsing employing distinct segmentation techniques, as described below.

- **Gold:** The pipeline accepts the gold segmentation from the Hebrew treebank.
- **Predicted:** The pipeline accepts the SOTA segmentation predicted by Seker and Tsarfaty (2020)'s segmention model.
- **Joint:** The joint scenario infers both the segmentation and the parse tree using the proposed architecture.

For each baseline we present SEG, POS and dependency parsing DEP scores.

Metrics The Labeled Attachment Score (LAS) serves as the predominant metric for measuring dependency parsing accuracy. However, this measurement method is incompatible for the complex segmentation task associated with Morphologically Rich Languages (MRLs) since the predicted segments (i.e., nodes in the tree) may differ from the gold ones. For this reason, we evaluate segmentation using the *aligned multi-set* $F_1 score$ (Seker and Tsarfaty (2020), Brusilovsky and Tsarfaty (2022)) metric, specifically chosen for to cope with cases where gold and predicted segmentations do not align, and also caters for backwards compatibility with previous work. All results we present are averaged over five distinct experiments with random seeds.

	SEG	**DEP**
Biaffine + Oracle SEG	100	86.76
Biaffine + Predicted SEG	97.6	71.57

Table 1: The original Biaffine architecture of Dozat and Manning (2017) with gold and predicted SEG.

Model →	**Stanza/Trankit**		
Input ↓	SEG	POS	DEP
Oracle SEG	100/100	94.75/97.2	78.38/89.42
Model SEG	89.51/95.2	85.03/92.68	67.45/83.55
Predicted SEG	97.6/97.6	92.73/94.92	75.52/**85.66**

Table 2: Trankit (Minh Nguyen and Nguyen, 2021) and Stanza (Qi et al., 2020) results for SEG, POS and DEP parsing in Hebrew. Oracle provides gold segments, Model provides the internal segmentation of Stanza/Trankit, and Predicted is the SOTA segmentation of AlephBERT (Seker and Tsarfaty, 2021).

6 Results and Analysis

Table 1 demonstrates the performance of the Dozat and Manning (2017) architecture using both gold *oracle* and predicted segmentation as input to the biaffine architecture. These results establish that when not using the gold (*oracle*) segmentation, even a small drop in segmentation leads to a substantial decline in dependency parsing accuracy, thereby emphasizing the importance of segmentation in parsing.

Table 2 then shows the results of Trankit (Minh Nguyen and Nguyen, 2021) and Stanza (Qi et al., 2020) compared with our proposed model.[5] Prior to this work, Trankit achieved state-of-the-art results on Hebrew parsing. The Oracle segmentation scenarios of Trankit and Stanza provide an idealized and unrealistic scenario, with a substantial drop when moving to non-gold scenarios. Notably, the experimental results of Trankit with our suggested external Hebrew segmentation sets a new SOTA to which our architecture achieves comparable results. The difference is minor, yet our proposed architecture stands out by offering an efficient full pipeline that delivers segmentation, tagging and parsing simultaneously, avoiding the

[5] Our models and code are publicly available at https://github.com/OnlpLab/Hebrew-Dependency-Parsing. All hyperparameters are listed in the Appendix.

	SEG	POS	DEP
No MTL	97.68	-	84.69
+ Gender	97.67	-	84.88
+ Person	97.61	-	84.99
+ Number	97.75	-	84.76
+ POS	**97.71**	94.41	**85.45**
+ POS (heBERT)	97.51	93.9	84.31
+ POS (mBERT)	96.84	91.8	80.68
All MTLs	97.68	94.31	85

Table 3: We evaluated our model under three different conditions: without employing any MTL, utilizing one MTL at a time, and incorporating all MTL components simultaneously.

	SEG	DEP
Static	96.36	78.19
Contextualized	97.18	82.23
Proposed	97.68	84.69

Table 4: Our model incorporates various embedding representations. The 'Static' and 'Contextualized' embeddings use a lattice context, 'Proposed' uses a valid sentence context for each analysis.

need to train, maintain, and install modules separately.

Table 3 shows the results of our proposed approach with ablation of the MTL contribution. These results demonstrate that our joint architecture surpasses the original Biaffine architecture in Hebrew parsing, attaining a state-of-the-art (SOTA) performance with an 85.45 $F_1 score$, better than the parsing frameworks of Stanza and Trankit. The results are comparable for the combination of Trankit with an external model with a separately trained decoder, with different training regimes, while in our model, SEG, POS and DEP are trained, and predicted, jointly.

Furthermore, Table 4 highlights the significance of the embedding method used for encoding the input lattices. While a substantial improvement is evident between static and contextualized embeddings, a notable enhancement is also observed when altering the context of the linearized lattice as we propose.

Table 5 illustrates the extent to which limitations of the MA component affect parsing performance, in cases where certain analyses may be absent for some tokens at inference time. It is evident that when the correct analyses are included in the set of possible analyses, it selects a better segmentation that results in more accurate parsing. So, improvement of the MA coverage is expected to yield even further improvement in parsing.

	SEG	POS	DEP
Infused	98.47	-	85.56
Infused + MTL POS	98.52	95.22	**86.55**
Uninfused	97.68	-	84.69
Uinfused + MTL POS	97.71	94.41	85.45

Table 5: Our proposed model with infused MA, with and without POS MTL.

Finally, since the LLM may be seamlessly replaced, further improvement may come from a better LLM encoder. Table 3 shows that replacing mBERT (Libovický et al., 2019) with the AlephBERT (Seker and Tsarfaty, 2021) encoder gave a significantly improved performance. This leaves a promise of further improving performance with significantly better LLMs.

Error Analysis We performed a manual error analysis on a subset of 50 sentences from the Hebrew UD HTB dev set. In these, there are merely 8 segmentation errors, with 5 of being a missing definite article (*'he hydia'h'*) and the remaining 3 involving incorrect segmentation of fused suffixes. In addition, a total of 108 dependency errors were identified, classified into four categories: prediction errors, wrong gold, truly ambiguous, and others (Table 7 in the Appendix). Of these, 70% are prediction errors. We categorized the errors based on the dependency labels that are involved. The predominant error type is associated with PP attachment, where 20% of the errors confuse the *obl* and *nmod* relations, indicating a confusion between the complements of the verb and modifiers of the noun, respectively (see further details in Table 8 in the Appendix).

7 Related and Future Work

Previous research has delved into lattice-based dependency parsing for MRLs such as Hebrew (More et al., 2019), Turkish (Seeker and Çetinoğlu, 2015), and Sanskrit (Krishna et al., 2020b). However, these prior contributions predominantly utilized graph-based and transition-based systems grounded in feature functions that are hand-engineered. In contrast, our current work takes a different perspective, presenting a purely neural architecture. A distinct challenge lies in generating embeddings for the lattice arcs, which represent a non-linear structure — an atypical input signal for language models. The aforementioned lattice-

based parsing architectures do not attend to this complexity thereby missing out on the advantages offered by contemporary Large Language Models (LLMs). This paper bypasses this divide, proposing an approach that effectively handles the intricate context and creates robust representations for lattices using neural encoders.

While neural studies in MRL parsing, such as the work by Sandhan et al. (2021), also leverage the Biaffine architecture of Dozat and Manning (2017), they typically focus on architectures that handles segmented and unambiguous inputs. Consequently, these models do not cope well with the challenges posed by the vast number of ambiguous words prevalent in MRLs such as Hebrew. In contrast, our proposed architecture accommodates ambiguous input, offering a unified solution that addresses both segmentation, morphological disambiguation, and parsing, in a single model.

In future research, we aim to assess our proposed framework on other MRLs, evaluate its performance across various language types and assess its generalization capabilities for lower-resourced languages. Additionally, we aim to explore further enhancements of MTL in parsing, by adding joint semantic predictions such as NER and SRL.

8 Conclusions

In this paper, we present a novel neural framework for jointly segmenting and parsing morpho-syntactic structures in Morphologically Rich Languages (MRLs). We address the intricate and complex nature of words in these languages and propose a method for incorporating linguistic information structured in a lattice into a neural parsing architecture. The contribution of this paper is manifold. First, we provide a language-agnostic neural joint architecture that can be used to confirm or disprove the joint hypothesis juxtaposed in the pre-neural era for MRLs. Second, we provide a thorough empirical investigation of Hebrew, providing SOTA results using a single joint model. Finally, as the proposed architecture relies on an LLM encoder, advances are expected to be achieved as LLMs further improve for low- and medium-resources MRLs, potentially closing the gap with non-MRLs.

9 Limitations

In our study, we introduce a joint morpho-syntactic architecture tailored to address the segmentation and parsing challenges of Morphologically Rich Languages (MRLs) in a single a model. It is important to note that the term “segmentation" can have various meanings, and in our work, we specifically refer to the segmentation of raw tokens into multiple meaning-bearing units, each of which carrying its own POS tag. This is compatible with previous work on Hebrew and other Semitic languages (Adler and Elhadad, 2006; Seker and Tsarfaty, 2021). All modeling and design decisions made are language-agnostic. Having acknowledged that, we conducted experiments using Hebrew as our test language. This investigation can and should be extended to any language that has a UD treebank and a wide-coverage morphological analyzer (MA).

One of the key components of our approach is the Morphological Analyzer (MA), which provides a list of possible analyses for each token. This component is not always freely available. Here, our experiments focused on Hebrew. It is noteworthy however that MAs are available for many languages and specifically for MRLs (More et al., 2018). MAs are available also for Arabic (Taji et al., 2018), Turkish (Yıldız et al., 2019) and Sanskrit.[6] It is also worth noting that the open MAs we can access is academia are fairly small, but there exist larger lexical MAs in the industry, for Hebrew and other languages.[7] On top of that, creating proper contextualized embeddings for each segment in the lattice is more time-consuming than is desired, and in future work we aim to specifically address these efficiency concerns.

Finally, when generating contextualized embeddings for the input lattice we employed AlephBert, a pre-trained monolingual language model for Hebrew. Substituting this model with a bigger or more advanced one could potentially yield further improvements. More work in the future may be done on improving the way we encode the linearized lattices, either in the realm of pre-tuning, or by fine-tuning the LLM specifically for the lattice-encoding task.

Acknowledgements

We thank Eylon Gueta, Refael Shaked Greenfeld and Amit Seker for fruitful discussions. We also thank the audience of the NLP-BIU seminar and three anonymous reviewers for thoughtful comments on earlier drafts. This research was funded

[6] http://sanskrit.jnu.ac.in/morph/analyze.jsp

[7] For instance through https://lexicala.com/

by the European Research Council (ERC) grant number 677352 and a Ministry of science and education (MOST) grant number 3-17992, for which we are grateful. The research was further supported by a grant from the Israeli Innovation Authority (KAMIN), and computing resources kindly funded by a VATAT grant and the Data Science Institute at Bar-Ilan University (BIU-DSI).

References

Meni Adler and Michael Elhadad. 2006. An unsupervised morpheme-based HMM for Hebrew morphological disambiguation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 665–672, Sydney, Australia. Association for Computational Linguistics.

Idan Brusilovsky and Reut Tsarfaty. 2022. Neural token segmentation for high token-internal complexity. In *arXiv:2203.10845v1*.

Shay B. Cohen and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 208–217, Prague, Czech Republic. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *In International Conference on Learning Representations (ICLR)*.

Michel Galley and Christopher D. Manning. 2009. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 773–781, Suntec, Singapore. Association for Computational Linguistics.

Aparna Garimella, Laura Chiticariu, and Yunyao Li. 2021. Domain-aware dependency parsing for questions. pages 4562–4568.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proceedings of ACL-08: HLT*, pages 371–379, Columbus, Ohio. Association for Computational Linguistics.

Spence Green and Christopher D. Manning. 2010. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 394–402, Beijing, China. Coling 2010 Organizing Committee.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, Sohee Yang, and Minjoon Seo. 2020. Spatial dependency parsing for 2d document understanding. *CoRR*, abs/2005.00642.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. In *Transactions of the Association for Computational Linguistics*, volume 4, pages 313–327.

Dan Kondratyuk and Milan Straka. 2019a. 75 languages, 1 model: Parsing universal dependencies universally. In *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLPIJCNLP)*, pages 2779–2795.

Dan Kondratyuk and Milan Straka. 2019b. 75 languages, 1 model: Parsing Universal Dependencies universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Amrith Krishna, Ashim Gupta, Deepak Garasangi, Pavankumar Satuluri, and Pawan Goyal. 2020a. Keep it surprisingly simple: A simple first order graph based parsing model for joint morphosyntactic parsing in Sanskrit. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4791–4797, Online. Association for Computational Linguistics.

Amrith Krishna, Bishal Santra, Ashim Gupta, Pavankumar Satuluri, and Pawan Goyal. 2020b. A graph-based framework for structured prediction tasks in Sanskrit. *Computational Linguistics*, 46(4):785–845.

Jindrich Libovický, Rudolf Rosa, and Alexander Fraser. 2019. How language-neutral is multilingual bert? volume abs/1911.03310.

Amir Pouran Ben Veyseh Minh Nguyen, Viet Lai and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, volume 322, pages 80–90.

Amir More, Amit Seker, Victoria Basmova, and Reut Tsarfaty. 2019. Joint transition-based models for morpho-syntactic parsing: Parsing strategies for MRLs and a case study from Modern Hebrew. volume 7, pages 33–48, Cambridge, MA. MIT Press.

Amir More, Özlem Çetinoğlu, Çagri Çöltekin, Nizar Habash, Benoît Sagot, Djamé Seddah, Dima Taji, and Reut Tsarfaty. 2018. Conll-ul: Universal morphological lattices for universal dependency parsing. In *International Conference on Language Resources and Evaluation*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108.

Jivnesh Sandhan, Amrith Krishna, Ashim Gupta, Laxmidhar Behera, and Pawan Goyal. 2021. A little pretraining goes a long way: A case study on dependency parsing task for low-resource morphologically rich languages. *arXiv preprint arXiv:2102.06551*.

Wolfgang Seeker and Özlem Çetinoğlu. 2015. A graph-based lattice dependency parser for joint morphological segmentation and syntactic analysis. volume 3, pages 359–373, Cambridge, MA. MIT Press.

Amit Seker and Reut Tsarfaty. 2020. A pointer network architecture for joint morphological segmentation and tagging. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4368–4378, Online. Association for Computational Linguistics.

Amit Seker and Reut Tsarfaty. 2021. Alephbert: A hebrew large pretrained language model to start-off your hebrew nlp application with. In *arXiv:2104.04052*.

Dima Taji, Salam Khalifa, Ossama Obeid, Fadhl Eryani, and Nizar Habash. 2018. An Arabic morphological analyzer and generator with copious features. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 140–150, Brussels, Belgium. Association for Computational Linguistics.

Reut Tsarfaty. 2006. Integrated morphological and syntactic disambiguation for Modern Hebrew. In *Proceedings of the COLING/ACL 2006 Student Research Workshop*, pages 49–54, Sydney, Australia. Association for Computational Linguistics.

Reut Tsarfaty, Dan Bareket, Stav Klein, and Amit Seker. 2020. From SPMRL to NMRL: What did we learn (and unlearn) in a decade of parsing morphologically-rich languages (MRLs)? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7396–7408, Online. Association for Computational Linguistics.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kuebler, Yannick Versley, Marie Candito, Jennifer Foster, Ines Rehbein, and Lamia Tounsi. 2010. Statistical parsing of morphologically rich languages (SPMRL) what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12, Los Angeles, CA, USA. Association for Computational Linguistics.

Olcay Taner Yıldız, Begüm Avar, and Gökhan Ercan. 2019. An open, extendible, and fast Turkish morphological analyzer. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1364–1372, Varna, Bulgaria. INCOMA Ltd.

A Appendix

A.1 Hyperparameters and Computing Resources

For all models we used the hyper parameters in Table 6. The research was conducted using a *NVIDIA GeForce GTX 1080 Ti* machine. To enhance time efficiency, we pre-generated embeddings before the training phase, avoiding the need to create them for each epoch. The process of generating embeddings for the entire training dataset took approximately 80 minutes. On average, each epoch lasted 15 seconds, resulting in a total training time of approximately 7 minutes.

For evaluation purposes, we assessed the efficiency of both embedding and inference on the test dataset, where the longest sentence consisted of 61 tokens with linearized lattice of 217 morphemes and the shortest contained 2 tokens with linearized lattice of 4 morphemes. The average linearized lattice contains 57 morphemes. The average time for embedding was 0.24 seconds, and for inference, it was 0.017 seconds. The maximum time recorded for embedding was 0.94 seconds, and for inference, it was 0.19 seconds. We acknowledge the efficiency bottleneck at the embedding generation phase, which we reserve for future research.

A.2 Error Analysis

We performed a manual error analysis by an expert on 50 sentences sampled from the dev set, and found 108 parsing errors. Table 7 presents the types of dependency errors, where 70% are prediction errors and the rest are not considered parser errors by the expert.

Embedding dimension	768
BiLSTM hidden size	600
Batch size	32
Embedding dropout	0.3
ARC MLP dropout	0.3
Label MLP dropout	0.3
All BiLSTMs depth	1
MLP depth	1
Arc MLP size	500
Label MLP size	100
Learning rate	0.001
MTL linear layer size	600

Table 6: Hyperparameter Settings

	number	**percent**
prediction error	76	70%
gold error	13	12%
ambiguous	11	10%
other	8	8%
all	108	100%

Table 7: Classification of errors by type.

Table 8 further presents the classification of errors by gold labels. For each label we count three types of errors: exclusively a head error, exclusively a label error, or an error encompassing both the head and label. We can see that *oblique* and *nmod* are top ranked, followed by *apposition, advmod* and *conj*. Interestingly, at the middle of the Table we see that on top of coordination *conj, cc*, which is known to be challenging to disambiguate, the construct-state construction *compound:smixut*, a well-known Semitic phenomenon, also appears to be confusing for the parser.

gold label	head	label	head + label	number	percent
obl	4	1	6	11	10.19%
nmod	7	2	1	10	9.26%
appos	4	1	4	9	8.34%
advmod	5	0	3	8	7.41%
conj	6	1	1	8	7.41%
cc	3	1	1	5	4.63%
ccomp	0	3	2	5	4.63%
compound:smixut	1	4	0	5	4.63%
dep	0	2	3	5	4.63%
amod	2	3	0	5	4.63%
acl:relcl	3	1	0	4	3.7%
case	3	0	1	4	3.7%
det	1	3	0	4	3.7%
obj	0	3	1	4	3.7%
nsubj	1	0	2	3	2.78%
nmod:poss	2	1	0	3	2.78%
fixed	1	0	2	3	2.78%
mark	2	1	0	3	2.78%
root	0	0	2	2	1.9%
advcl	1	1	0	2	1.9%
acl	0	0	2	2	1.9%
parataxis	0	1	0	1	0.93%
xcomp	0	0	1	1	0.93%
flat:name	0	1	0	1	0.93%
Total	46	30	32	108	100%

Table 8: Classification of errors by gold labels. Each label is divided into three types of errors: exclusively a head error, exclusively a label error, or an error encompassing both the head and label.