**UDPipe 1.1 and 1.2**

**Tokenization**
- UD utilizes three level tokenization: sentences, tokens and words
- sentences and tokens breaks are predicted jointly using trainable bidirectional GRU recurrent network
- if the token breaks are not marked in the data, they can be generated automatically utilizing arbitrary plain text in given language
- tokens are split into words by rules obtained from the training data
  - full token rules
  - token suffix rules
  - fast inference (direct C++ implementation, caching in the network)
- UDPIpe 1.1 changes
  - spaces in tokens are allowed (but only if present in data)
  - paragraph and document boundaries are both produced and used during training (not to predict sentence breaks at the end)
  - allow perfect reconstruction of spaces
- **Morphological Analysis**
  - two guessers generating from a suffix (and sometimes also prefix)
  - UFPOSTAG & XUPOSTAG & FEATS triples
  - UFPOSTAG & LEMMA pairs
  - lemmas generated either absolutely or relatively to the word form

**POS Tagging and Lemmatization**
- old-school averaged perceptron with Viterbi order 3 decoding

**Parsing**
- transitional-based parser based on simple NN classifier
- transition systems for non-projective and projective trees
- novel search-based oracle usable with any transition system
- dynamic oracle for the projective system
- very fast, but unfortunately not SoTA performance

**CoNLL 2017 UD Shared Task**

**UDPipe 1.1 Baseline System**
- tokenizer, POS tagger, lemmatizer and parser provided to the participants of the CoNLL 2017 UD Shared Task
- trained without using development data
- preprocessed test set available during TIRA evaluation
- for surprise languages, jackknifed POS tags on test sets also provided
- 13th out of 33 contestent systems

**UDPipe 1.2 Participant System**
- full training data used, with hyperparameter search on dev data
- slightly larger hyperparameters
  - tokenizer GRU dimension (24→64), form embeddings (50→64)
- merging treebanks of the same language
- enriching training data of small treebanks utilizing other treebanks of the same language
- append only 14, 15, 1 or 2 times the size of the original treebank due to often inconsistent annotation in different treebanks
- joint segmentation and parsing
- difficult segmentation in treebanks with missing punctuation
- choose segmentation maximizing dependency trees logprob
- 8th out of 33 contestent systems

**Search-based Oracle Evaluation**
- the overall effect of search-based oracle on various transition systems across all UD 2.0 treebanks

A 2.0 treebanks

<table>
<thead>
<tr>
<th>Transition system and oracle</th>
<th>No search-based oracle</th>
<th>Search-based oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>LAS</td>
<td>CAS</td>
</tr>
<tr>
<td>Are standard system with static oracle</td>
<td>74.29</td>
<td>66.27</td>
</tr>
<tr>
<td>Are standard system with dynamic oracle</td>
<td>75.31</td>
<td>69.36</td>
</tr>
<tr>
<td>Swap system with static lazy oracle</td>
<td>74.73</td>
<td>66.76</td>
</tr>
<tr>
<td>Link system with static oracle</td>
<td>74.79</td>
<td>66.78</td>
</tr>
<tr>
<td>Any system, static oracle</td>
<td>74.72</td>
<td>68.71</td>
</tr>
<tr>
<td>Any system, any oracle</td>
<td>75.27</td>
<td>69.31</td>
</tr>
</tbody>
</table>

**Future Work**
- more accurate neural network models
  - deeper, RNNS, biaffine, both pretrained&learned embeddings, ...
  - add multithreaded and GPU support for both training and inference

**Model Size and Runtime Performance**
- displayed as a median for all UD 2.0 treebanks, together with the 5th and 95th percentile.
- complete model consists of a tokenizer with dimension 64, a tagger, a lemmatizer and a parser with beam size 5

<table>
<thead>
<tr>
<th>Model configuration</th>
<th>Model size [MB]</th>
<th>Model speed [words/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokenizer dim 24</td>
<td>0.04 (0.03–0.15)</td>
<td>27.7 (20–37)</td>
</tr>
<tr>
<td>Tokenizer dim 64</td>
<td>0.20 (0.19–0.31)</td>
<td>6.0 (4.9–8.6)</td>
</tr>
<tr>
<td>Tagger&amp;lemmatizer</td>
<td>9.4 (2.3–24.8)</td>
<td>6.5 (2.1–14)</td>
</tr>
<tr>
<td>Parser beam size 1</td>
<td>3.2 (1.9–6.9)</td>
<td>2.7 (2.2–3.6)</td>
</tr>
<tr>
<td>Parser beam size 5</td>
<td>13.2 (4.4–31.9)</td>
<td>1.7 (1.2–2.3)</td>
</tr>
</tbody>
</table>

**UDPipe Implementation**
- easily trainable from CoNLL-U file
- hyperparameter search support
- CC BY-SA-NC models for UD 2.0 treebanks (50 languages)
- efficient standalone C++ implementation under MPL
- precompiled Linux/Windows/OS X binaries
- Python package on PyPI
- Perl package on CPAN, precompiled Java and C# binding
- REST service (both running and a provided server), web application

**Demo and Download:** [http://ufal.mff.cuni.cz/udpipe](http://ufal.mff.cuni.cz/udpipe)

**Acknowledgements:** This work has been partially supported and has been published in tools developed and stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2015071), also partially supported by OP VVV projects CZ.02.1.01/0.0/0.0/16_013/0001781 and CZ.02.2.69/0.0/0.0/16_018/0002573, and by SVV project number 260 453.