trankit

NLP Group at the University of Oregon

Oct 15, 2021

INTRODUCTION

News: Trankit v1.0.0 is out. The new version has pretrained pipelines using XLM-Roberta Large which is much better than our previous version with XLM-Roberta Base. Checkout the performance comparison here. Trankit v1.0.0 also provides a brand new command-line interface that helps users who are not familiar with Python can use Trankit more easily. Finally, the new version has a brand new Auto Mode in which a language detector is used to activate the language-specific models for processing the input, avoiding switching back and forth between languages in a multilingual pipeline.

Trankit is a *light-weight Transformer-based Python* Toolkit for multilingual Natural Language Processing (NLP). It provides a trainable pipeline for fundamental NLP tasks over 100 languages, and 90 pretrained pipelines for 56 languages. Built on a state-of-the-art pretrained language model, Trankit significantly outperforms prior multilingual NLP pipelines over sentence segmentation, part-of-speech tagging, morphological feature tagging, and dependency parsing while maintaining competitive performance for tokenization, multi-word token expansion, and lemmatization over 90 Universal Dependencies v2.5 treebanks. Our pipeline also obtains competitive or better named entity recognition (NER) performance compared to existing popular toolkits on 11 public NER datasets over 8 languages.

Trankit's Github Repo is available at: https://github.com/nlp-uoregon/trankit

Trankit's Demo Website is hosted at: http://nlp.uoregon.edu/trankit

CHAPTER

ONE

CITATION

If you use Trankit in your research or software. Please cite our following paper:

```
@inproceedings{nguyen2021trankit,
   title={Trankit: A Light-Weight Transformer-based Toolkit for Multilingual Natural␣
↪Language Processing},
   author={Nguyen, Minh Van and Lai, Viet and Veyseh, Amir Pouran Ben and Nguyen, Thien␣
↪Huu},
   booktitle="Proceedings of the 16th Conference of the European Chapter of the␣
↪Association for Computational Linguistics: System Demonstrations",
   year={2021}
}
```

1.1 News: Trankit v1.0.0 is out

What's new in Trankit v1.0.0? Let's check out below.

1.1.1 Installation

```
pip install trankit==1.1.0
```

1.1.2 Trankit large

Starting from version 1.0.0, Trankit supports using XLM-Roberta large as the multilingual embedding (i.e., Trankit large), which further boosts the performance over 90 Universal Dependencies treebanks. The usage of the large version is the same as before except that users need to specify the embedding for pipeline initialization. Below is an example for initializing a monolingual and multilingual pipeline.

```
from trankit import Pipeline

# Initialize an English pipeline with XLM-Roberta large
p = Pipeline('english', embedding='xlm-roberta-large')

# Initialize a multilingual pipeline for ['english', 'chinese', 'arabic'] with XLM-Roberta␣
↪large
p = Pipeline('english', embedding='xlm-roberta-large')
```

(continues on next page)

(continued from previous page)

```
p.add('chinese')
p.add('arabic')
```

Currently, the argument `embedding` can only be set to one of the two values `['xlm-roberta-large', 'xlm-roberta-base']`. If the argument `embedding` is not specifically set, Trankit will use `'xlm-roberta-base'` for its multilingual embedding by default.

1.1.3 Auto mode for multilingual pipelines

Starting from version v1.0.0, Trankit supports a handy *Auto Mode* in which users do not have to set a particular language active before processing the input. In the Auto Mode, Trankit will automatically detect the language of the input and use the corresponding language-specific models, thus avoiding switching back and forth between languages in a multilingual pipeline. Specifically, there are two methods to turn on the Auto Mode.

The first method is to initialize a multilingual pipeline with a special code `'auto'`. After the initialization, the pipeline would be automatically set in the Auto Mode.

```
from trankit import Pipeline

p = Pipeline('auto')

# Tokenizing an English input
en_output = p.tokenize('''I figured I would put it out there anyways.''')

# POS, Morphological tagging and Dependency parsing a French input
fr_output = p.posdep('''On pourra toujours parler à propos d'Averroès de "décentrement␣
↪du Sujet".''')

# NER tagging a Vietnamese input
vi_output = p.ner('''Cuc tiêm th nghim tin hành ti Hc vin Quân y, Hà Ni''')
```

Note that, the multilingual pipeline in this case is initialized with all supported languages and all these languages would be considered for the language detection process. In some cases where we want to constrain the detected language to belong to a specific set, the second method is used:

```
from trankit import Pipeline

p = Pipeline('english')
p.add('french')
p.add('vietnamese')
p.set_auto(True)

# Tokenizing an English input
en_output = p.tokenize('''I figured I would put it out there anyways.''')

# POS, Morphological tagging and Dependency parsing a French input
fr_output = p.posdep('''On pourra toujours parler à propos d'Averroès de "décentrement␣
↪du Sujet".''')

# NER tagging a Vietnamese input
vi_output = p.ner('''Cuc tiêm th nghim tin hành ti Hc vin Quân y, Hà Ni''')
```

In this way, we are guaranteed that the detected language can only be one of the added languages `["english", "french", "vietnamese"]`. Suppose that at some point later, we want to turn off the Auto Mode, this can be done easily with a single line of code as follows:

```
p.set_auto(False)
```

After this, our multilingual pipeline can be used in the manual mode where we can manually set a particular language active. As a final note, we use langid to perform language detection. The detected language for each input can be inspected by accessing the field `"lang"` of the output. Thank you loretoparisi for your suggestion on this.

1.1.4 Command-line interface

Starting from version v1.0.0, Trankit supports processing text via command-line interface. This helps users who are not familiar with Python programming language can use Trankit more easily. Please check out this page for tutorials and examples.

1.2 Installation

Installing *Trankit* is easily done via one of the following methods:

1.2.1 Using pip

```
pip install trankit
```

The command would install *Trankit* and all dependent packages automatically.

1.2.2 From source

```
git clone https://github.com/nlp-uoregon/trankit
cd trankit
pip install -e .
```

This would first clone our github repo and install Trankit.

1.2.3 Fixing the compatibility issue of Trankit with Transformers

Previous versions of Trankit have encountered the compatibility issue when using recent versions of transformers. To fix this issue, please install the new version of Trankit as follows:

```
pip install trankit==1.1.0
```

If you encounter any other problem with the installation, please raise an issue here to let us know. Thanks.

1.3 Quick examples

1.3.1 Initialize a pipeline

Monolingual usage

Before using any function of Trankit, we need to initialize a pipeline. Here is how we can do it for English:

```
from trankit import Pipeline

p = Pipeline('english')
```

In this example, *Trankit* receives the string `'english'` specifying which language package it should use to initialize a pipeline. To know which language packages are supported we can check this table or directly print out the attribute `trankit.supported_langs`:

```
import trankit

print(trankit.supported_langs)
# Output: ['afrikaans', 'ancient-greek-perseus', 'ancient-greek', 'arabic', 'armenian', 'basque',
↪ 'belarusian', 'bulgarian', 'catalan', 'chinese', 'traditional-chinese', 'classical-chinese',
↪'croatian', 'czech-cac', 'czech-cltt', 'czech-fictree', 'czech', 'danish', 'dutch', 'dutch-
↪lassysmall', 'english', 'english-gum', 'english-lines', 'english-partut', 'estonian',
↪'estonian-ewt', 'finnish-ftb', 'finnish', 'french', 'french-partut', 'french-sequoia', 'french-
↪spoken', 'galician', 'galician-treegal', 'german', 'german-hdt', 'greek', 'hebrew', 'hindi',
↪'hungarian', 'indonesian', 'irish', 'italian', 'italian-partut', 'italian-postwita', 'italian-
↪twittiro', 'italian-vit', 'japanese', 'kazakh', 'korean', 'korean-kaist', 'kurmanji', 'latin',
↪'latin-perseus', 'latin-proiel', 'latvian', 'lithuanian', 'lithuanian-hse', 'marathi',
↪'norwegian-nynorsk', 'norwegian-nynorsklia', 'norwegian-bokmaal', 'old-french', 'old-russian',
↪ 'persian', 'polish-lfg', 'polish', 'portuguese', 'portuguese-gsd', 'romanian-nonstandard',
↪'romanian', 'russian-gsd', 'russian', 'russian-taiga', 'scottish-gaelic', 'serbian', 'slovak',
↪'slovenian', 'slovenian-sst', 'spanish', 'spanish-gsd', 'swedish-lines', 'swedish', 'tamil',
↪'telugu', 'turkish', 'ukrainian', 'urdu', 'uyghur', 'vietnamese']
```

By default, *trankit* would try to use GPU if a GPU device is available. However, we can force it to run on CPU by setting the tag `gpu=False`:

```
from trankit import Pipeline

p = Pipeline('english', gpu=False)
```

Another tag that we can use is `cache_dir`. By default, *Trankit* would check if the pretrained model files exist. If they don't, it would download all pretrained files including the shared XLMR-related files and the separate language-related files, then store them to `./cache/trankit`. However, we can change this by setting the tag `cache_dir`:

```
from trankit import Pipeline

p = Pipeline('english', cache_dir='./path-to-your-desired-location/')
```

Multilingual usage

Processing multilingual inputs is easy and effective with *Trankit*. For example, to initilize a pipeline that can process inputs of the 3 languages English, Chinese, and Arabic, we can do as follows:

```
from trankit import Pipeline

p = Pipeline('english')
p.add('chinese')
p.add('arabic')
```

Each time the add function is called for a particular language (e.g., `'chinese'` and `'arabic'` in this case), *Trankit* would only download the language-related files. Therefore, the downloading would be very fast. Here is what will show up when the above snippet is executed:

```
from trankit import Pipeline

p = Pipeline('english')
# Output:
# Downloading: 100%|| 5.07M/5.07M [00:00<00:00, 9.28MB/s]
# http://nlp.uoregon.edu/download/trankit/english.zip
# Downloading: 100%|| 47.9M/47.9M [00:00<00:00, 89.2MiB/s]
# Loading pretrained XLM-Roberta, this may take a while...
# Downloading: 100%|| 512/512 [00:00<00:00, 330kB/s]
# Downloading: 100%|| 1.12G/1.12G [00:14<00:00, 74.8MB/s]
# Loading tokenizer for english
# Loading tagger for english
# Loading lemmatizer for english
# Loading NER tagger for english
# ===================================================
# Active language: english
# ===================================================

p.add('chinese')
# http://nlp.uoregon.edu/download/trankit/chinese.zip
# Downloading: 100%|| 40.4M/40.4M [00:00<00:00, 81.3MiB/s]
# Loading tokenizer for chinese
# Loading tagger for chinese
# Loading lemmatizer for chinese
# Loading NER tagger for chinese
# ===================================================
# Added languages: ['english', 'chinese']
# Active language: english
# ===================================================

p.add('arabic')
# http://nlp.uoregon.edu/download/trankit/arabic.zip
# Downloading: 100%|| 38.6M/38.6M [00:00<00:00, 76.8MiB/s]
# Loading tokenizer for arabic
# Loading tagger for arabic
# Loading multi-word expander for arabic
# Loading lemmatizer for arabic
# Loading NER tagger for arabic
# ===================================================
```

(continues on next page)

(continued from previous page)

```
# Added languages: ['english', 'chinese', 'arabic']
# Active language: english
# ================================================
```

As we can see, each time a new language is added, the list of the added languages inreases. However, the active langage remains the same, i.e., `'english'`. This indicates that the pipeline can work with inputs of the 3 specified languages, however, it is assuming that the inputs that it will receive are in `'english'`. To change this assumption, we need to "tell" the pipeline that we're going to process inputs of a particular language, for example:

```
p.set_active('chinese')
# ================================================
# Active language: chinese
# ================================================
```

From now, the pipeline is ready to process `'chinese'` inputs. To make sure that the language is activated successfully, we can access the attribute `active_lang` of the pipeline:

```
print(p.active_lang)
# 'chinese'
```

1.3.2 Document-level processing

The following lines of code show the basic use of *Trankit* with English inputs.

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
doc_text = '''Hello! This is Trankit.'''

# perform all tasks on the input
all = p(doc_text)
```

Here, `doc_text` is assumed to be a document. Then, the sentence segmentation and tokenization are then jointly done. For each sentence, *Trankit* performs part-of-speech tagging, morphological feature tagging, dependency parsing, and also named entity recognition (NER) if the pretrained NER model for that language is available. The result of the entire process is stored in the variable `all`, which is a hierarchical native Python dictionary that we can retrieve different types of information at both the document and sentence level. The output would look like this (we use [...] to improve the visualization):

```
{
  'text': 'Hello! This is Trankit.',  # input string
  'sentences': [ # list of sentences
    {
      'id': 1, 'text': 'Hello!', 'dspan': (0, 6), 'tokens': [...]
    },
    {
      'id': 2,  # sentence index
      'text': 'This is Trankit.',  'dspan': (7, 23), # sentence span
```

(continues on next page)

(continued from previous page)

```
      'tokens': [ # list of tokens
        {
          'id': 1, # token index
          'text': 'This', 'upos': 'PRON', 'xpos': 'DT',
          'feats': 'Number=Sing|PronType=Dem',
          'head': 3, 'deprel': 'nsubj', 'lemma': 'this', 'ner': 'O',
          'dspan': (7, 11), # document-level span of the token
          'span': (0, 4) # sentence-level span of the token
        },
        {'id': 2...},
        {'id': 3...},
        {'id': 4...}
      ]
    }
  ]
}
```

Below we show some examples for accessing different information of the output.

Text

At document level, we have two fields to access, which are `'text'` storing the input string and `'sentences'` storing the tagged sentences of the input. Suppose that we want to get the text form of the first sentence, we would first access the field `'sentences'` of the output to get the list of sentences, use an index to locate the sentence in the list, then finally access the `'text'` field:

```
sent_text = all['sentences'][1]['text']
print(sent_text)
# Output: This is Trankit.
```

Span

we can also get the text form of the sentence manually with the `'dspan'` field that provides the span-based location of the sentence in the document:

```
dspan = all['sentences'][1]['dspan']
print(dspan)
# Output: (7, 23)
sent_text = doc_text[dspan[0]: dspan[1]]
print(sent_text)
# Output: This is Trankit.
```

Note that, we use `'dspan'` with the prefix `d` to indicate that this information is at document level.

Token list

Each sentence is associated with a list of tokens, which can be accessed via the `'tokens'` field. Each token is in turn a dictionary with different types of information. For example, we can get the information of the first token of the second sentence as follows:

```
token = all['sentences'][1]['tokens'][0]
print(token)
```

The information of the token is stored in a Python dictionary:

```
{
  'id': 1,                    # token index
  'text': 'This',             # text form of the token
  'upos': 'PRON',             # UPOS tag of the token
  'xpos': 'DT',               # XPOS tag of the token
  'feats': 'Number=Sing|PronType=Dem',    # morphological feature of the token
  'head': 3,                  # index of the head token
  'deprel': 'nsubj',          # dependency relation between from the current token and␣
↪the head token
  'dspan': (7, 11),           # document-level span of the token
  'span': (0, 4),             # sentence-level span of the token
  'lemma': 'this',            # lemma of the token
  'ner': 'O'              # Named Entity Recognitation (NER) tag of the token
}
```

Here, we provide two different types of span for each token: `'dspan'` and `'span'`. `'dspan'` is used for the global location of the token in the document while `'span'` provides the local location of the token in the sentence. We can use either one of these two fields to manually retrieve the text form of the token like this:

```
# retrieve the text form via 'dspan'
dspan = token['dspan']
print(doc_text[dspan[0]: dspan[1]])
# Output: This

# retrieve the text form via 'span'
span = token['span']
print(sent_text[span[0]: span[1]])
# Output: This
```

1.3.3 Sentence-level processing

In many cases, we may want to use *Trankit* to process a sentence instead of a document. This can be achieved by setting the tag `is_sent=True`:

```
sent_text = '''Hello! This is Trankit.'''
tokens = p(sent_text, is_sent=True)
```

The output is now a dictionary with a list of all tokens, instead of a list of sentences as before.

```
{
  'text': 'Hello! This is Trankit.',
  'tokens': [
```

(continues on next page)

(continued from previous page)

```
    {
      'id': 1,
      'text': 'Hello',
      'upos': 'INTJ',
      'xpos': 'UH',
      'head': 5,
      'deprel': 'discourse',
      'lemma': 'hello',
      'ner': 'O',
      'span': (0, 5)
    },
    {'id': 2...},
    {'id': 3...},
    {'id': 4...},
    {'id': 5...},
    {'id': 6...},
  ]
}
```

For more examples on other functions, please refer to the following sections: *Sentence Segmentation*, *Tokenization*, *Part-of-speech, Morphological tagging and Dependency parsing*, *Lemmatization*, *Named entity recognition*, and *Building a customized pipeline*.

1.4 How Trankit works

Note: The best way to understand how Trankit works is to look at our technical paper, which is available at: https://arxiv.org/pdf/2101.03289.pdf

In this section, we briefly present the most important details of the technologies used by Trankit.

Natural Language Processing (NLP) pipelines of current state-of-the-art multilingual NLP Toolkits such as UDPipe (Straka, 2018) and Stanza (Qi et al., 2020) are trained separately and do not share any component, especially the embedding layers that account for most of the model size. This makes their memory usage grow aggressively as pipelines for more languages are simultaneously needed and loaded into the memory. Most importantly, *these toolkits have not explored contextualized embeddings from pretrained transformer-based language models* that have the potentials to significantly improve the performance of the NLP tasks, as demonstrated in many prior works (Devlin et al., 2018, Liu et al., 2019b, Conneau et al., 2020). This motivates us to develop Trankit that can overcome such limitations.

First, we utilize the state-of-the-art multilingual pretrained transformer XLM-Roberta (Conneau et al., 2020) to build three components: the joint token and sentence splitter; the joint model for part-of-speech, morphological tagging, and dependency parsing; and the named entity recognizer (See the figure above). As a result, our system advances state-of-theart performance for sentence segmentation, part-of-speech (POS) tagging, morphological feature tagging, and dependency parsing while achieving competitive or better performance for tokenization, multi-word token expansion, and lemmatizatio over the 90 treebanks.

Second, we simultaneously solve the problem of loading pipelines for many languages into the memory and the problem of the transformer size with our novel plug-and-play mechanism with Adapters (Pfeiffer et al., 2020a, Pfeiffer et al., 2020b). In particular, a set of adapters (for transfomer layers) and task-specific weights (for final predictions) are created for each transformer-based component for each language while only one single large multilingual pretrained transformer is shared across components and languages. During training, the shared pretrained transformer is fixed while only the adapters and task-specific weights are updated. At inference time, depending on the language of the input text and the current active component, the corresponding trained adapter and task-specific weights are activated and plugged into

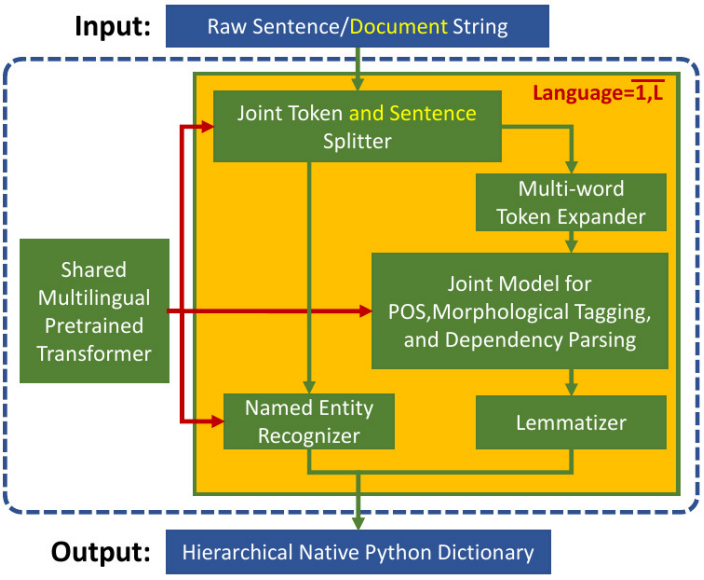

Fig. 1: Overall architecture of Trankit. Among the five models, three (i.e., joint token and sentence splitter; joint model for part-of-speech tagging, morphological tagging, and dependency parsing; and named entity recognizer) in Trankit are transformer-based. They all share a single multilingual pretrained transformer.

the pipeline to process the input. This mechanism not only solves the memory problem but also substantially reduces the training time.

1.5 Model performance

News: Trankit v1.0.0 is out. The new version has pretrained pipelines using XLM-Roberta Large which is much better than our previous version with XLM-Roberta Base. Checkout the performance comparison below on this page. Trankit v1.0.0 also provides a brand new command-line interface that helps users who are not familiar with Python can use Trankit more easily. Finally, the new version has a brand new Auto Mode in which a language detector is used to activate the language-specific models for processing the input, avoiding switching back and forth between languages in a multilingual pipeline.

1.5.1 Universal Dependencies v2.5

The following table shows the performance comparison between Trankit large (using XLM-Roberta large) and base (using XLM-Roberta base), spaCy v2.3, UDPipe v1.2, and Stanza v1.1.1 on the test treebanks of the 5 major languages on the Universal Dependencies v2.5 corpora. Performance for each model are F1 scores obtained by using the official evaluation script of the CoNLL 2018 Shared Task. The meanings of the columns in the following tables are provided by the evaluation script as follows:

- **Tokens**: how well do the gold tokens match system tokens.
- **Sents.**: how well do the gold sentences match system sentences.
- **Words**: how well can the gold words be aligned to system words.
- **UPOS**: using aligned words, how well does UPOS match.
- **XPOS**: using aligned words, how well does XPOS match.
- **UFeats**: using aligned words, how well does universal FEATS match.

- **AllTags**: using aligned words, how well does UPOS+XPOS+FEATS match.
- **Lemmas**: using aligned words, how well does LEMMA match.
- **UAS**: using aligned words, how well does HEAD match.
- **LAS**: using aligned words, how well does HEAD+DEPREL(ignoring subtypes) match.
- **CLAS**: using aligned words with content DEPREL, how well does HEAD+DEPREL(ignoring subtypes) match.
- **MLAS**: using aligned words with content DEPREL, how well does HEAD+DEPREL(ignoring subtypes)+UPOS+UFEATS+FunctionalChildren(DEPREL+UPOS+UFEATS) match.
- **BLEX**: using aligned words with content DEPREL, how well does HEAD+DEPREL(ignoring subtypes)+LEMMAS match.

Treebank	System	Tokens	Sents.	Words	UPOS	XPOS	UFeats	Lemmas	UAS	LAS
Arabic-PADT	Trankit[large]	99.95	96.79	99.39	96.8	94.93	94.99	94.92	89.74	85.98
	Trankit[base]	99.93	96.59	99.22	96.31	94.08	94.28	94.65	88.39	84.68
	Stanza	99.98	80.43	97.88	94.89	91.75	91.86	93.27	83.27	79.33
	UDPipe	99.98	82.09	94.58	90.36	84.00	84.16	88.46	72.67	68.14
Chinese-GSD	Trankit[large]	97.75	99.4	97.75	95.13	94.94	97.28	97.75	87.38	84.82
	Trankit[base]	97.01	99.7	97.01	94.21	94.02	96.59	97.01	85.19	82.54
	Stanza	92.83	98.80	92.83	89.12	88.93	92.11	92.83	72.88	69.82
	UDPipe	90.27	99.10	90.27	84.13	84.04	89.05	90.26	61.60	57.81
English-EWT	Trankit[large]	98.67	90.49	98.67	96.65	96.47	96.75	97.25	91.29	89.4
	Trankit[base]	98.48	88.35	98.48	95.95	95.71	96.26	96.84	90.14	87.96
	Stanza	99.01	81.13	99.01	95.40	95.12	96.11	97.21	86.22	83.59
	UDPipe	98.90	77.40	98.90	93.26	92.75	94.23	95.45	80.22	77.03
	spaCy	97.30	61.19	97.30	86.72	90.83	•	87.05	•	•
French-GSD	Trankit[large]	99.77	98.8	99.72	97.89	•	97.34	97.9	94.97	93.37
	Trankit[base]	99.7	96.63	99.66	97.85	•	97.16	97.80	94.00	92.34
	Stanza	99.68	94.92	99.48	97.30	•	96.72	97.64	91.38	89.05
	UDPipe	99.68	93.59	98.81	95.85	•	95.55	96.61	87.14	84.26
	spaCy	98.34	77.30	94.15	86.82	•	•	87.29	67.46	60.60
Spanish-Ancora	Trankit[large]	99.91	99.39	99.91	99.06	99	98.85	99.15	94.77	93.29
	Trankit[base]	99.94	99.13	99.93	99.02	98.94	98.80	99.17	94.11	92.41
	Stanza	99.98	99.07	99.98	98.78	98.67	98.59	99.19	92.21	90.01
	UDPipe	99.97	98.32	99.95	98.32	98.13	98.13	98.48	88.22	85.10
	spaCy	99.47	97.59	98.95	94.04	•	•	79.63	86.63	84.13

As can be seen from the table, both the large and base version of Trankit outperform other toolkits over different tasks

(e.g., POS and morphological tagging) in which the improvement boost is substantial and significant for sentence segmentation and dependency parsing. For example, **English** enjoys a **9.36%** improvement for **sentence segmentation**, a **5.07%** and **5.81%** improvement for UAS and LAS in **dependency parsing**. For **Arabic**, Transit has a remarkable improvement of **16.36%** for **sentence segmentation** while **Chinese** observes **14.50%** and **15.0%** improvement of UAS and LAS for **dependency parsing**.

Next, we show the detailed performance comparison of Trankit large (using XLM-Roberta large), Trankit base (using XLM-Roberta base), and Stanza v1.1.1 on 90 Universal Dependencies v2.5 treebanks. Over all 90 treebanks, both versions of Trankit outperform the previous state-of-the-art framework Stanza in most of the tasks, particularly for sentence segmentation (+4.28%), POS tagging (+2.00% for UPOS and +2.14% for XPOS), morphological tagging (+2.18%), and dependency parsing (+5.61% for UAS and +6.81% for LAS) while maintaining the competitive performance on tokenization, multi-word token expansion, and lemmatization.

		Tokens	Sentences	Words	UPOS	XPOS	UFeats	A
Overall performance	Trankit[large]	99.32	92.86	99.15	96.22	94.64	93.93	9
	Trankit[base]	99.23	91.82	99.02	95.65	94.05	93.21	9
	Stanza	99.26	88.58	98.90	94.21	92.50	91.75	8
	Trankit[large] - Trankit[base]	0.09	1.04	0.13	0.56	0.59	0.72	0
	Trankit[large] - Stanza	0.07	4.28	0.26	2.00	2.14	2.18	2
Afrikaans-AfriBooms	Trankit[large]	99.84	100.00	99.84	98.87	96.63	98.68	9
	Trankit[base]	99.72	100.00	99.72	98.55	96.10	98.26	9
	Stanza	99.75	99.65	99.75	97.56	94.27	97.03	9
Ancient_Greek-Perseus	Trankit[large]	99.71	98.70	99.71	93.97	87.25	91.66	8
	Trankit[base]	99.66	98.70	99.66	92.72	84.89	89.95	8
	Stanza	99.98	98.85	99.98	92.54	85.22	91.06	8
Ancient_Greek-PROIEL	Trankit[large]	99.91	67.60	99.91	97.86	97.93	93.03	9
	Trankit[base]	99.83	58.62	99.83	97.30	97.37	91.59	9
	Stanza	100.00	51.65	100.00	97.38	97.75	92.09	9
Arabic-PADT	Trankit[large]	99.95	96.79	99.39	96.80	94.93	94.99	9
	Trankit[base]	99.93	96.59	99.22	96.31	94.08	94.28	9
	Stanza	99.98	80.43	97.88	94.89	91.75	91.86	9
Armenian-ArmTDP	Trankit[large]	98.52	99.46	97.26	95.03	97.26	90.42	8
	Trankit[base]	98.41	98.92	97.23	94.36	97.23	88.89	8
	Stanza	98.96	95.22	96.58	92.49	96.58	88.19	8
Basque-BDT	Trankit[large]	99.89	100.00	99.89	96.95	99.89	93.96	9
	Trankit[base]	99.81	99.92	99.81	96.23	99.81	92.35	9
	Stanza	100.00	100.00	100.00	96.23	100.00	93.09	9
Belarusian-HSE	Trankit[large]	99.81	82.28	99.81	96.46	31.77	82.82	2
	Trankit[base]	99.53	84.68	99.53	91.58	36.18	78.47	2
	Stanza	99.38	78.24	99.38	91.92	31.34	77.73	2
Bulgarian-BTB	Trankit[large]	99.78	98.79	99.78	99.29	97.75	98.30	9
	Trankit[base]	99.84	98.25	99.84	99.18	97.42	98.10	9
	Stanza	99.93	97.27	99.93	98.68	96.35	97.59	9
Catalan-AnCora	Trankit[large]	99.94	99.76	99.94	99.11	99.07	98.66	9
	Trankit[base]	99.94	100.00	99.93	99.02	98.97	98.57	9
	Stanza	99.99	99.84	99.98	98.75	98.66	98.29	9
Chinese-GSD	Trankit[large]	97.75	99.40	97.75	95.13	94.94	97.28	9
	Trankit[base]	97.01	99.70	97.01	94.21	94.02	96.59	9
	Stanza	92.83	98.80	92.83	89.12	88.93	92.11	8
Classical_Chinese-Kyoto	Trankit[large]	99.70	70.58	99.70	92.89	91.90	94.43	8
	Trankit[base]	99.63	61.82	99.63	92.07	91.03	93.88	8
	Stanza	99.47	46.95	99.47	90.25	89.64	92.68	8

Table 1 – continued from previous page

Croatian-SET	Trankit[large]	99.93	99.08	99.93	98.58	96.55	96.85	9
	Trankit[base]	99.92	99.16	99.92	98.38	96.08	96.52	9
	Stanza	99.96	98.15	99.96	97.88	94.86	95.32	9
Czech-CAC	Trankit[large]	99.99	100.00	99.98	99.64	98.17	98.01	9
	Trankit[base]	99.96	100.00	99.95	99.42	97.40	97.12	9
	Stanza	99.99	100.00	99.97	98.76	94.79	93.52	9
Czech-CLTT	Trankit[large]	99.89	98.89	99.85	99.17	93.80	94.02	9
	Trankit[base]	99.82	100.00	99.76	98.93	93.36	93.68	9
	Stanza	99.93	100.00	99.84	98.92	91.89	91.97	9
Czech-FicTree	Trankit[large]	99.98	99.50	99.98	99.11	97.26	97.99	9
	Trankit[base]	99.97	99.38	99.97	98.94	96.47	97.09	9
	Stanza	99.97	98.60	99.96	98.31	95.23	96.01	9
Czech-PDT	Trankit[large]	99.95	97.87	99.95	99.32	98.19	98.19	9
	Trankit[base]	99.94	97.85	99.94	99.23	97.81	97.77	9
	Stanza	99.97	94.14	99.97	98.50	95.38	94.61	9
Danish-DDT	Trankit[large]	99.81	95.46	99.81	98.71	99.81	98.33	9
	Trankit[base]	99.79	95.19	99.79	98.35	99.79	97.79	9
	Stanza	99.96	93.57	99.96	97.75	99.96	97.38	9
Dutch-Alpino	Trankit[large]	99.43	90.65	99.43	96.67	95.01	96.55	9
	Trankit[base]	99.22	89.88	99.22	96.55	94.92	96.22	9
	Stanza	99.96	89.98	99.96	96.33	94.76	96.28	9
Dutch-LassySmall	Trankit[large]	99.36	92.60	99.36	96.52	95.57	96.63	9
	Trankit[base]	99.21	91.09	99.21	96.20	95.18	96.25	9
	Stanza	99.90	77.95	99.90	95.97	94.87	96.22	9
English-EWT	Trankit[large]	98.67	90.49	98.67	96.65	96.47	96.75	9
	Trankit[base]	98.48	88.35	98.48	95.95	95.71	96.26	9
	Stanza	99.01	81.13	99.01	95.40	95.12	96.11	9
English-GUM	Trankit[large]	99.52	91.60	99.52	96.66	96.51	97.47	9
	Trankit[base]	99.45	91.63	99.45	96.39	96.24	97.19	9
	Stanza	99.82	86.35	99.82	95.89	95.91	96.87	9
English-LinES	Trankit[large]	99.46	91.87	99.46	97.31	95.91	96.96	9
	Trankit[base]	99.53	93.01	99.53	97.14	95.54	96.67	9
	Stanza	99.95	88.49	99.95	96.88	95.18	96.76	9
English-ParTUT	Trankit[large]	99.71	100.00	99.65	96.86	96.65	95.77	9
	Trankit[base]	99.66	100.00	99.60	96.79	96.55	95.94	9
	Stanza	99.68	100.00	99.59	96.15	95.83	95.21	9
Estonian-EDT	Trankit[large]	99.75	96.58	99.75	97.87	98.35	97.10	9
	Trankit[base]	99.72	96.55	99.72	97.53	98.13	96.56	9
	Stanza	99.96	93.32	99.96	97.19	98.04	95.77	9
Estonian-EWT	Trankit[large]	97.76	82.58	97.76	94.26	94.93	91.49	8
	Trankit[base]	96.96	83.72	96.96	92.07	93.16	89.17	8
	Stanza	99.20	67.14	99.20	88.86	91.70	87.16	8
Finnish-FTB	Trankit[large]	99.84	97.36	99.83	98.32	97.29	98.09	9
	Trankit[base]	99.75	95.83	99.74	97.46	96.23	97.22	9
	Stanza	100.00	89.59	99.97	95.50	95.12	96.51	9
Finnish-TDT	Trankit[large]	99.71	97.22	99.72	98.48	98.78	96.84	9
	Trankit[base]	99.62	95.98	99.62	97.99	98.44	96.52	9
	Stanza	99.77	93.05	99.73	96.97	97.72	95.36	9
French-GSD	Trankit[large]	99.77	98.80	99.72	97.89	99.72	97.34	9
	Trankit[base]	99.70	96.63	99.66	97.85	99.66	97.16	9

Table 1 – continued from previous page

	Stanza	99.68	94.92	99.48	97.30	99.47	96.72	9
French-ParTUT	Trankit[large]	99.76	98.63	99.65	97.66	97.35	94.55	9
	Trankit[base]	99.74	98.63	99.69	97.77	97.54	94.20	9
	Stanza	99.82	100.00	99.37	96.60	96.37	93.98	9
French-Sequoia	Trankit[large]	99.81	94.07	99.78	99.22	99.78	98.43	9
	Trankit[base]	99.73	94.36	99.73	98.90	99.73	97.98	9
	Stanza	99.90	88.79	99.58	98.19	99.58	97.58	9
French-Spoken	Trankit[large]	99.36	53.06	99.19	96.80	96.91	99.19	9
	Trankit[base]	99.38	39.39	99.18	96.73	96.73	99.18	9
	Stanza	100.00	22.09	99.45	95.49	97.06	99.45	9
Galician-CTG	Trankit[large]	99.76	98.44	99.31	97.30	97.05	99.17	9
	Trankit[base]	99.76	98.09	99.38	97.17	96.83	99.23	9
	Stanza	99.89	99.13	99.32	97.21	96.99	99.14	9
Galician-TreeGal	Trankit[large]	99.47	95.52	99.22	97.62	95.68	96.50	9
	Trankit[base]	99.47	94.60	99.06	97.06	94.90	95.89	9
	Stanza	99.59	89.17	98.41	94.29	91.81	93.36	9
German-GSD	Trankit[large]	99.71	89.72	99.72	95.23	97.68	91.68	8
	Trankit[base]	99.75	92.72	99.75	95.04	97.57	91.51	8
	Stanza	99.53	85.79	99.53	94.07	96.98	89.52	8
German-HDT	Trankit[large]	99.92	99.67	99.92	98.44	98.41	94.05	9
	Trankit[base]	99.90	99.50	99.90	98.42	98.37	93.95	9
	Stanza	100.00	97.41	100.00	98.04	97.94	91.77	9
Greek-GDT	Trankit[large]	99.85	93.50	99.85	98.41	98.41	96.34	9
	Trankit[base]	99.75	93.57	99.75	98.04	98.04	95.41	9
	Stanza	99.88	93.18	99.89	97.84	97.84	94.94	9
Hebrew-HTB	Trankit[large]	99.81	99.69	96.31	94.32	94.32	93.03	9
	Trankit[base]	99.79	100.00	96.03	93.75	93.75	91.96	9
	Stanza	99.98	99.69	93.19	90.46	90.46	89.24	8
Hindi-HDTB	Trankit[large]	99.88	99.91	99.88	98.01	97.70	93.91	9
	Trankit[base]	99.89	99.64	99.89	97.77	97.38	94.03	9
	Stanza	100.00	99.44	100.00	97.59	97.08	94.03	9
Hungarian-Szeged	Trankit[large]	99.59	99.33	99.59	97.49	99.59	95.23	9
	Trankit[base]	99.41	98.00	99.41	96.97	99.41	94.47	9
	Stanza	99.87	97.00	99.87	96.03	99.87	93.76	9
Indonesian-GSD	Trankit[large]	99.89	95.54	99.89	93.39	95.06	96.11	8
	Trankit[base]	99.86	95.37	99.86	93.57	94.18	95.67	8
	Stanza	99.99	93.78	99.99	93.68	94.79	96.00	8
Irish-IDT	Trankit[large]	99.47	98.24	99.47	94.72	93.74	80.90	7
	Trankit[base]	99.32	97.25	99.32	93.88	92.46	80.36	7
	Stanza	99.76	95.93	99.76	93.90	92.43	78.19	7
Italian-ISDT	Trankit[large]	99.88	99.07	99.86	98.72	98.63	98.32	9
	Trankit[base]	99.88	98.76	99.87	98.58	98.46	98.20	9
	Stanza	99.91	98.76	99.76	98.01	97.91	97.72	9
Italian-ParTUT	Trankit[large]	99.81	100.00	99.79	98.58	98.42	98.15	9
	Trankit[base]	99.82	100.00	99.81	98.63	98.41	98.16	9
	Stanza	99.81	100.00	99.77	97.82	97.76	97.79	9
Italian-PoSTWITA	Trankit[large]	99.34	73.95	99.18	96.60	96.43	96.52	9
	Trankit[base]	99.29	69.95	99.07	96.10	95.91	95.87	9
	Stanza	99.71	63.70	99.46	96.19	96.04	96.28	9
Italian-TWITTIRO	Trankit[large]	99.15	65.72	98.89	95.47	94.90	94.09	9

Table 1 – continued from previous page

	Trankit[base]	99.22	56.00	99.01	95.31	94.74	93.83	9
	Stanza	99.34	52.40	98.76	94.41	94.01	93.34	9
Italian-VIT	Trankit[large]	99.97	98.18	99.84	98.07	97.29	97.76	9
	Trankit[base]	99.99	96.52	99.81	97.82	97.02	97.39	9
	Stanza	99.98	94.92	99.49	97.21	96.23	96.79	9
Japanese-GSD	Trankit[large]	95.25	95.88	95.25	93.66	93.47	95.23	9
	Trankit[base]	94.57	95.49	94.57	92.86	92.44	94.56	9
	Stanza	92.67	94.57	92.67	91.16	90.84	92.66	9
Kazakh-KTB	Trankit[large]	95.98	81.71	95.37	77.94	77.47	63.01	5
	Trankit[base]	94.48	90.00	93.62	75.94	75.67	62.28	5
	Stanza	93.46	88.56	94.16	56.23	56.10	42.73	3
Korean-GSD	Trankit[large]	98.57	98.08	98.57	95.71	90.88	98.35	8
	Trankit[base]	98.63	97.67	98.63	95.63	90.32	98.43	8
	Stanza	99.88	96.65	99.88	96.18	90.14	99.66	8
Korean-Kaist	Trankit[large]	98.70	99.87	98.70	95.13	88.07	98.70	8
	Trankit[base]	98.79	99.14	98.79	94.99	87.62	98.79	8
	Stanza	100.00	99.93	100.00	95.45	86.31	100.00	8
Kurmanji-MG	Trankit[large]	94.95	91.50	94.63	75.07	74.16	57.15	5
	Trankit[base]	94.52	80.56	94.20	74.33	73.44	56.54	5
	Stanza	94.81	87.43	94.49	57.17	55.91	43.02	3
Latin-ITTB	Trankit[large]	100.00	94.54	100.00	98.97	97.29	97.98	9
	Trankit[base]	100.00	94.57	100.00	98.76	96.74	97.54	9
	Stanza	99.99	80.66	99.99	98.09	95.38	96.43	9
Latin-Perseus	Trankit[large]	99.60	97.93	99.60	92.84	83.33	86.79	8
	Trankit[base]	99.45	97.87	99.45	90.15	77.12	81.12	7
	Stanza	100.00	98.24	100.00	90.63	78.42	82.42	7
Latin-PROIEL	Trankit[large]	99.85	66.10	99.85	97.79	97.75	93.22	9
	Trankit[base]	99.82	58.16	99.82	96.80	96.83	91.28	9
	Stanza	100.00	43.04	100.00	96.92	97.10	91.24	9
Latvian-LVTB	Trankit[large]	99.73	98.69	99.73	97.61	91.22	95.18	9
	Trankit[base]	99.71	99.10	99.71	97.16	90.24	94.47	8
	Stanza	99.82	99.01	99.82	96.03	88.25	93.46	8
Lithuanian-ALKSNIS	Trankit[large]	99.84	95.72	99.84	97.45	93.98	94.46	9
	Trankit[base]	99.82	95.10	99.82	97.03	92.35	93.00	9
	Stanza	99.87	88.79	99.87	93.37	85.67	87.84	8
Lithuanian-HSE	Trankit[large]	97.71	100.00	97.71	90.59	89.85	79.64	7
	Trankit[base]	98.22	94.55	98.22	90.46	89.71	77.92	7
	Stanza	97.53	51.11	97.53	81.08	80.04	70.72	6
Marathi-UFAL	Trankit[large]	99.20	69.31	97.22	87.79	97.22	70.62	6
	Trankit[base]	99.20	60.87	95.25	82.83	95.25	69.43	6
	Stanza	98.00	76.40	92.25	77.24	92.25	60.27	5
Norwegian_Nynorsk-Nynorsk	Trankit[large]	99.84	98.97	99.84	98.52	99.84	97.79	9
	Trankit[base]	99.81	98.71	99.81	98.20	99.81	97.20	9
	Stanza	99.97	94.85	99.97	97.92	99.97	96.88	9
Norwegian_Nynorsk-NynorskLIA	Trankit[large]	99.76	99.53	99.76	96.48	99.76	95.59	9
	Trankit[base]	99.74	99.53	99.74	96.31	99.74	95.41	9
	Stanza	100.00	99.69	100.00	95.92	100.00	94.82	9
Norwegian-Bokmaal	Trankit[large]	99.88	98.89	99.88	98.85	99.88	98.07	9
	Trankit[base]	99.88	99.20	99.88	98.66	99.88	97.60	9
	Stanza	99.99	97.17	99.99	98.29	99.99	97.17	9

Table 1 – continued from previous page

Old_French-SRCMF	Trankit[large]	99.91	100.00	99.91	96.96	96.83	98.32	9
	Trankit[base]	99.84	100.00	99.84	96.36	96.21	97.75	9
	Stanza	100.00	100.00	100.00	96.05	96.09	97.74	9
Old_Russian-TOROT	Trankit[large]	98.87	51.91	98.87	94.70	94.63	89.61	8
	Trankit[base]	98.44	42.22	98.44	92.63	92.66	86.75	8
	Stanza	100.00	35.69	100.00	93.63	93.83	86.76	8
Persian-Seraji	Trankit[large]	99.26	99.25	99.20	97.78	97.67	97.70	9
	Trankit[base]	99.22	99.25	99.11	97.35	97.24	97.36	9
	Stanza	100.00	99.25	99.65	97.29	97.30	97.37	9
Polish-LFG	Trankit[large]	98.34	99.57	98.34	97.84	95.52	96.00	9
	Trankit[base]	98.32	99.91	98.32	97.66	94.59	95.05	9
	Stanza	99.95	99.83	99.95	98.55	94.66	95.84	9
Polish-PDB	Trankit[large]	99.93	98.71	99.92	99.16	96.92	97.11	9
	Trankit[base]	99.91	98.53	99.89	99.06	96.29	96.44	9
	Stanza	99.87	98.39	99.83	98.31	94.04	94.27	9
Portuguese-Bosque	Trankit[large]	99.75	97.18	99.67	97.52	99.67	96.50	9
	Trankit[base]	99.70	97.48	99.65	97.27	99.65	96.50	9
	Stanza	99.77	94.30	99.67	97.04	99.67	96.36	9
Portuguese-GSD	Trankit[large]	99.81	97.10	99.72	98.43	98.43	99.61	9
	Trankit[base]	99.82	96.76	99.71	98.30	98.30	99.61	9
	Stanza	99.96	98.00	99.87	98.18	98.18	99.79	9
Romanian-Nonstandard	Trankit[large]	98.74	98.00	98.74	96.23	91.64	90.51	8
	Trankit[base]	98.68	98.57	98.68	96.04	91.48	90.33	8
	Stanza	98.96	97.53	98.96	95.40	90.73	89.79	8
Romanian-RRT	Trankit[large]	99.60	98.49	99.60	97.90	97.35	97.43	9
	Trankit[base]	99.72	97.67	99.72	97.87	97.25	97.44	9
	Stanza	99.77	96.64	99.77	97.54	96.97	97.13	9
Russian-GSD	Trankit[large]	99.79	99.25	99.79	98.25	97.89	95.78	9
	Trankit[base]	99.63	98.25	99.63	97.96	97.65	94.86	9
	Stanza	99.65	97.16	99.65	97.38	97.18	93.11	9
Russian-SynTagRus	Trankit[large]	99.71	99.45	99.71	99.06	99.71	98.20	9
	Trankit[base]	99.71	99.14	99.71	98.94	99.71	97.85	9
	Stanza	99.57	98.86	99.57	98.20	99.57	95.91	9
Russian-Taiga	Trankit[large]	98.90	92.32	98.90	96.16	96.62	91.02	8
	Trankit[base]	98.77	92.60	98.77	95.50	97.27	89.42	8
	Stanza	97.11	85.79	97.11	92.25	94.70	85.76	8
Scottish_Gaelic-ARCOSG	Trankit[large]	99.43	57.46	99.42	94.70	88.65	90.71	8
	Trankit[base]	99.26	54.10	99.25	92.98	85.47	88.25	8
	Stanza	99.48	55.35	99.47	92.50	84.89	87.99	8
Serbian-SET	Trankit[large]	99.91	100.00	99.91	99.06	96.22	96.40	9
	Trankit[base]	99.91	99.71	99.91	98.97	95.82	95.96	9
	Stanza	100.00	99.33	100.00	98.44	94.26	94.55	9
Simplified_Chinese-GSDSimp	Trankit[large]	97.66	98.20	97.66	94.99	94.78	97.24	9
	Trankit[base]	96.94	99.70	96.94	94.17	93.98	96.51	9
	Stanza	92.92	99.10	92.92	89.05	88.84	92.12	8
Slovak-SNK	Trankit[large]	99.94	98.49	99.94	97.90	90.82	95.22	9
	Trankit[base]	99.93	98.07	99.93	97.80	89.02	94.00	8
	Stanza	99.97	90.93	99.97	96.34	87.15	91.59	8
Slovenian-SSJ	Trankit[large]	99.97	100.00	99.97	99.24	97.83	97.99	9
	Trankit[base]	99.93	99.81	99.93	99.03	96.70	96.97	9

Table 1 – continued from previous page

	Stanza	99.91	91.60	99.91	98.29	95.08	95.37	9
Slovenian-SST	Trankit[large]	99.84	33.03	99.84	95.81	91.74	91.86	8
	Trankit[base]	99.79	31.96	99.79	94.90	90.27	90.37	8
	Stanza	100.00	26.59	100.00	93.66	88.09	88.06	8
Spanish-AnCora	Trankit[large]	99.91	99.39	99.91	99.06	99.00	98.85	9
	Trankit[base]	99.94	99.13	99.93	99.02	98.94	98.80	9
	Stanza	99.98	99.07	99.98	98.78	98.67	98.59	9
Spanish-GSD	Trankit[large]	99.93	99.30	99.88	97.38	99.88	96.96	9
	Trankit[base]	99.91	98.94	99.88	97.41	99.88	96.88	9
	Stanza	99.96	95.97	99.87	96.69	99.87	96.40	9
Swedish-LinES	Trankit[large]	99.89	90.64	99.89	98.05	95.77	90.98	8
	Trankit[base]	99.73	90.57	99.73	97.60	95.23	90.50	8
	Stanza	99.94	86.99	99.94	96.97	94.58	90.11	8
Swedish-Talbanken	Trankit[large]	99.91	99.26	99.91	99.06	98.09	98.04	9
	Trankit[base]	99.87	99.38	99.87	98.76	97.77	97.73	9
	Stanza	99.97	98.85	99.97	97.65	96.57	96.70	9
Tamil-TTB	Trankit[large]	98.33	100.00	94.44	87.64	83.92	87.19	8
	Trankit[base]	98.02	100.00	93.64	86.18	82.09	86.43	8
	Stanza	99.58	95.08	91.42	82.60	78.80	81.89	7
Telugu-MTG	Trankit[large]	98.89	98.62	98.89	94.31	94.31	98.33	9
	Trankit[base]	98.89	98.62	98.89	94.32	94.32	97.92	9
	Stanza	100.00	97.95	100.00	92.93	92.93	99.17	9
Turkish-IMST	Trankit[large]	99.84	98.32	98.91	96.20	95.33	92.78	9
	Trankit[base]	99.86	98.18	98.68	95.15	94.35	92.02	8
	Stanza	99.89	97.62	98.07	94.21	93.43	92.08	9
Ukrainian-IU	Trankit[large]	99.77	97.55	99.76	98.50	96.32	96.09	9
	Trankit[base]	99.78	97.72	99.76	98.33	94.96	94.94	9
	Stanza	99.81	96.65	99.79	96.77	92.49	92.53	9
Urdu-UDTB	Trankit[large]	99.75	97.67	99.75	94.43	92.77	81.91	7
	Trankit[base]	99.66	98.32	99.66	94.15	92.66	83.04	7
	Stanza	100.00	98.88	100.00	94.42	92.62	84.21	8
Uyghur-UDT	Trankit[large]	97.95	88.95	97.95	88.50	91.34	86.56	7
	Trankit[base]	97.63	88.51	97.63	87.47	90.37	85.31	7
	Stanza	99.79	86.90	99.79	89.45	91.92	87.92	8
Vietnamese-VTB	Trankit[large]	94.88	96.63	94.88	89.70	88.14	94.64	8
	Trankit[base]	95.22	96.25	95.22	89.40	87.85	95.03	8
	Stanza	87.25	93.15	87.25	79.50	77.90	87.02	7

Performance for Stanza, UDPipe, and spaCy is obtained using their public pretrained models. The overall performance for Trankit and Stanza is computed as the macro-averaged F1 over 90 treebanks using the official evaluation script of the CoNLL 2018 Shared Task.

1.5.2 Named Entity Recognition

Performance comparison between Trankit large, Trankit base, and Stanza v1.1.1 on the test sets of 11 public NER datasets. Performance is based on entity micro-averaged F1.

Language	Corpus	Trankit[large]	Trankit[base]	Stanza v1.1.1
Arabic	AQMAR	76.1	74.8	74.3
Chinese	OntoNotes	80.5	80	79.2
Dutch	CoNLL02	93.8	91.8	89.2
	WikiNER	95.0	94.8	94.8
English	CoNLL03	92.5	92.1	92.1
	OntoNotes	90.2	89.6	88.8
French	WikiNER	92.9	92.3	92.9
German	CoNLL03	85.3	84.6	81.9
	GermEval14	89.4	86.9	85.2
Russian	WikiNER	93.2	92.8	92.9
Spanish	CoNLL02	89.2	88.9	88.1

1.6 Supported Languages

1.6.1 Trainable Languages

In the table below we show 100 languages that users can use the training data of such languages to train their own pipelines with Trankit.

Trainable Languages			
Afrikaans	Estonian	Kyrgyz	Sindhi
Albanian	Filipino	Lao	Sinhala
Amharic	Finnish	Latin	Slovak
Arabic	French	Latvian	Slovenian
Armenian	Galician	Lithuanian	Somali
Assamese	Georgian	Macedonian	Spanish
Azerbaijani	German	Malagasy	Sundanese
Basque	Greek	Malay	Swahili
Belarusian	Gujarati	Malayalam	Swedish
Bengali	Hausa	Marathi	Tamil
Bengali	Hebrew	Mongolian	Tamil Romanized
Bosnian	Hindi	Nepali	Telugu
Breton	Hindi Romanized	Norwegian	Telugu Romanized
Bulgarian	Hungarian	Oriya	Thai
Burmese	Icelandic	Oromo	Turkish
Burmese	Indonesian	Pashto	Ukrainian
Catalan	Irish	Persian	Urdu
Chinese (Simplified)	Italian	Polish	Urdu Romanized
Chinese (Traditional)	Japanese	Portuguese	Uyghur
Croatian	Javanese	Punjabi	Uzbek
Czech	Kannada	Romanian	Vietnamese
Danish	Kazakh	Russian	Welsh
Dutch	Khmer	Sanskrit	Western Frisian
English	Korean	Scottish Gaelic	Xhosa
Esperanto	Kurdish (Kurmanji)	Serbian	Yiddish

1.6.2 Pretrained Languages & Their Code Names

Trankit provides 90 pretrained pipelines for 56 languages. Each pretrained pipeline is associated with a treebank that it is trained on. Below we show the 56 pretrained languages, their corresponding treebanks, and the code names to initialize pretrained pipelines. *The pretrained pipelines can be directly downloaded by clicking on their code names in the table below.*

Note that, the names of the default treebanks are put inside the brackets []. For example, English has 4 treebanks, which are *UD_English-EWT*, *UD_English-GUM*, *UD_English-LinES*, and *UD_English-ParTUT*. The treebank *UD_English-EWT* is put inside a bracket [], so it is the default treebank for English. Looking at the following table, we can select the appropriate code name and follow the instructions here to initialize a pipeline. For example, if we want to initialize a pipeline that is trained on the default treebank *UD_English-EWT*, we can use the code name *english*; to initialize a pipeline that is trained on a non-default treebank such as *UD_English-GUM*, we can use *english-gum*.

Language	Treebank	Code Name (for pipeline initialization)	Requires MWT expansio
Afrikaans	[UD_Afrikaans-AfriBooms]	afrikaans	
Ancient Greek	[UD_Ancient_Greek-PROIEL]	ancient-greek	
	UD_Ancient_Greek-Perseus	ancient-greek-perseus	
Arabic	[UD_Arabic-PADT]	arabic	Yes
Armenian	[UD_Armenian-ArmTDP]	armenian	Yes
Basque	[UD_Basque-BDT]	basque	
Belarusian	[UD_Belarusian-HSE]	belarusian	
Bulgarian	[UD_Bulgarian-BTB]	bulgarian	

Catalan	[UD_Catalan-AnCora]	catalan	Yes
Chinese (simplified)	[UD_Simplified_Chinese-GSDSimp]	chinese	
Chinese (traditional)	[UD_Chinese-GSD]	traditional-chinese	
Chinese (classical)	[UD_Classical_Chinese-Kyoto]	classical-chinese	
Croatian	[UD_Croatian-SET]	croatian	
Czech	UD_Czech-CAC	czech-cac	Yes
	UD_Czech-CLTT	czech-cltt	Yes
	UD_Czech-FicTree	czech-fictree	Yes
	[UD_Czech-PDT]	czech	Yes
Danish	[UD_Danish-DDT]	danish	
Dutch	[UD_Dutch-Alpino]	dutch	
	UD_Dutch-LassySmall	dutch-lassysmall	
English	[UD_English-EWT]	english	
	UD_English-GUM	english-gum	
	UD_English-LinES	english-lines	
	UD_English-ParTUT	english-partut	Yes
Estonian	[UD_Estonian-EDT]	estonian	
	UD_Estonian-EWT	estonian-ewt	
Finnish	UD_Finnish-FTB	finnish-ftb	Yes
	[UD_Finnish-TDT]	finnish	Yes
French	[UD_French-GSD]	french	Yes
	UD_French-ParTUT	french-partut	Yes
	UD_French-Sequoia	french-sequoia	Yes
	UD_French-Spoken	french-spoken	Yes
Galician	[UD_Galician-CTG]	galician	Yes
	UD_Galician-TreeGal	galician-treegal	Yes
German	[UD_German-GSD]	german	Yes
	UD_German-HDT	german-hdt	
Greek	[UD_Greek-GDT]	greek	Yes
Hebrew	[UD_Hebrew-HTB]	hebrew	Yes
Hindi	[UD_Hindi-HDTB]	hindi	
Hungarian	[UD_Hungarian-Szeged]	hungarian	
Indonesian	[UD_Indonesian-GSD]	indonesian	
Irish	[UD_Irish-IDT]	irish	
Italian	[UD_Italian-ISDT]	italian	Yes
	UD_Italian-ParTUT	italian-partut	Yes
	UD_Italian-PoSTWITA	italian-postwita	Yes
	UD_Italian-TWITTIRO	italian-twittiro	Yes
	UD_Italian-VIT	italian-vit	Yes
Japanese	[UD_Japanese-GSD]	japanese	
Kazakh	[UD_Kazakh-KTB]	kazakh	Yes
Korean	[UD_Korean-GSD]	korean	
	UD_Korean-Kaist	korean-kaist	
Kurmanji	[UD_Kurmanji-MG]	kurmanji	
Latin	[UD_Latin-ITTB]	latin	
	UD_Latin-Perseus	latin-perseus	
	UD_Latin-PROIEL	latin-proiel	
Latvian	[UD_Latvian-LVTB]	latvian	
Lithuanian	[UD_Lithuanian-ALKSNIS]	lithuanian	
	UD_Lithuanian-HSE	lithuanian-hse	
Marathi	[UD_Marathi-UFAL]	marathi	Yes

Norwegian (Bokmaal)	[UD_Norwegian-Bokmaal]	norwegian-bokmaal	
Norwegian (Nynorsk)	[UD_Norwegian_Nynorsk-Nynorsk]	norwegian-nynorsk	
	UD_Norwegian_Nynorsk-NynorskLIA	norwegian-nynorsklia	
Old French	[UD_Old_French-SRCMF]	old-french	
Old Russian	[UD_Old_Russian-TOROT]	old-russian	
Persian	[UD_Persian-Seraji]	persian	Yes
Polish	UD_Polish-LFG	polish-lfg	
	[UD_Polish-PDB]	polish	Yes
Portuguese	[UD_Portuguese-Bosque]	portuguese	Yes
	UD_Portuguese-GSD	portuguese-gsd	Yes
Romanian	UD_Romanian-Nonstandard	romanian-nonstandard	
	[UD_Romanian-RRT]	romanian	
Russian	UD_Russian-GSD	russian-gsd	
	[UD_Russian-SynTagRus]	russian	
	UD_Russian-Taiga	russian-taiga	
Scottish Gaelic	[UD_Scottish_Gaelic-ARCOSG]	scottish-gaelic	
Serbian	[UD_Serbian-SET]	serbian	
Slovak	[UD_Slovak-SNK]	slovak	
Slovenian	[UD_Slovenian-SSJ]	slovenian	
	UD_Slovenian-SST	slovenian-sst	
Spanish	[UD_Spanish-AnCora]	spanish	Yes
	UD_Spanish-GSD	spanish-gsd	Yes
Swedish	UD_Swedish-LinES	swedish-lines	
	[UD_Swedish-Talbanken]	swedish	
Tamil	[UD_Tamil-TTB]	tamil	Yes
Telugu	[UD_Telugu-MTG]	telugu	
Turkish	[UD_Turkish-IMST]	turkish	Yes
Ukrainian	[UD_Ukrainian-IU]	ukrainian	Yes
Urdu	[UD_Urdu-UDTB]	urdu	
Uyghur	[UD_Uyghur-UDT]	uyghur	
Vietnamese	[VLSP + UD_Vietnamese-VTB]	vietnamese	
	UD_Vietnamese-VTB	vietnamese-vtb	

1.7 Sentence segmentation

NOTE: *Quick examples* might be helpful for using this function.

The sample code for performing sentence segmentation on a raw text is:

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
doc_text = '''Hello! This is Trankit.'''

sentences = p.ssplit(doc_text)

print(sentences)
```

The output of the sentence segmentation module is a native Python dictionary with a list of the split sentences. For each sentence, we can access its span which is handy for retrieving the sentnece's location in the original document. The output would look like this:

```
{
  'text': 'Hello! This is Trankit.',
  'sentences': [
    {
      'id': 1,
      'text': 'Hello!',
      'dspan': (0, 6)
    },
    {
      'id': 2,
      'text': 'This is Trankit.',
      'dspan': (7, 23)
    }
  ]
}
```

1.8 Tokenization

NOTE: *Quick examples* might be helpful for using this function.

Tokenization module of *Trankit* can work with both sentence-level and document-level inputs.

1.8.1 Document-level tokenization

For document inputs, *trankit* first performs tokenization and sentence segmentation jointly to obtain a list of tokenized sentences. Below is how we can use this function:

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
doc_text = '''Hello! This is Trankit.'''

tokenized_doc = p.tokenize(doc_text)

print(tokenized_doc)
```

The returned `tokenized_doc` should look like this:

```
{
  'text': 'Hello! This is Trankit.',
  'sentences': [
    {
      'id': 1,
      'text': 'Hello!', 'dspan': (0, 6),
```

(continues on next page)

(continued from previous page)

```
      'tokens': [
        {
          'id': 1,
          'text': 'Hello',
          'dspan': (0, 5),
          'span': (0, 5)
        },
        {
          'id': 2,
          'text': '!',
          'dspan': (5, 6),
          'span': (5, 6)
        }
      ]
    },
    {
      'id': 2,
      'text': 'This is Trankit.', 'dspan': (7, 23)
      'tokens': [
        {
          'id': 1,
          'text': 'This',
          'dspan': (7, 11),
          'span': (0, 4)
        },
        {
          'id': 2,
          'text': 'is',
          'dspan': (12, 14),
          'span': (5, 7)
        },
        {
          'id': 3,
          'text': 'Trankit',
          'dspan': (15, 22),
          'span': (8, 15)
        },
        {
          'id': 4,
          'text': '.',
          'dspan': (22, 23),
          'span': (15, 16)
        }
      ]
    }
  ]
}
```

For each sentence, *trankit* provides the information of its location in the document via `'dspan'` field. For each token, there are two types of span that we can access: (i) Document-level span (via `'dspan'`) and (ii) Sentence-level span (via `'span'`). Check *this* to know how these fields work.

1.8.2 Sentence-level tokenization

In some cases, we might already have the sentences and only want to do tokenization for each sentence. This can be achieved by setting the tag `is_sent=True` when we call the function `.tokenize()`:

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
sent_text = '''This is Trankit.'''

tokens = p.tokenize(sent_text, is_sent=True)

print(tokens)
```

This will return a list of tokens. The output will look like this:

```
{
  'text': 'This is Trankit.',
  'tokens': [
    {
      'id': 1,
      'text': 'This',
      'span': (0, 4)
    },
    {
      'id': 2,
      'text': 'is',
      'span': (5, 7)
    },
    {
      'id': 3,
      'text': 'Trankit',
      'span': (8, 15)
    },
    {
      'id': 4,
      'text': '.',
      'span': (15, 16)
    }
  ]
}
```

As the input is assumed to be a sentence, we only have the sentence-level span for each token.

1.8.3 Multi-word token expansion

In addition to tokenization, some languages also require *Multi-word token expansion*. That means, each token can be expanded into multiple syntactic words. This process is helpful for these languages when performing later tasks such as part-of-speech, morphological tagging, dependency parsing, and lemmatization. Below is an example for such case in French:

```
from trankit import Pipeline

p = Pipeline('french')

doc_text = '''Je sens qu'entre ça et les films de médecins et scientifiques fous que nous
avons déjà vus, nous pourrions emprunter un autre chemin pour l'origine. On
pourra toujours parler à propos d'Averroès de "décentrement du Sujet".'''

out_doc = p.tokenize(doc_text)
print(out_doc['sentences'][1])
```

For illustration purpose, we only show part of the second sentence:

```
{
    'text': 'Je sens qu\'entre ça et les films de médecins et scientifiques fous que␣
↪nous\navons déjà vus, nous pourrions emprunter un autre chemin pour l\'origine. On\
↪npourra toujours parler à propos d\'Averroès de "décentrement du Sujet".',
    'sentences': [
      ...
      ,
      {
        'id': 2,
        'text': 'On\npourra toujours parler à propos d\'Averroès de "décentrement du␣
↪Sujet".',
        'dspan': (149, 222),
        'tokens': [
          ...
          ,
          {
            'id': 11,
            'text': 'décentrement',
            'dspan': (199, 211),
            'span': (50, 62)
          },
          {
            'id': (12, 13), # token index
            'text': 'du', # text form
            'expanded': [ # list of syntactic words
              {
                'id': 12, # token index
                'text': 'de' # text form
              },
              {
                'id': 13, # token index
                'text': 'le' # text form
              }
            ],
```

(continues on next page)

(continued from previous page)

```
            'span': (63, 65),
            'dspan': (212, 214)
          },
          {
            'id': 14,
            'text': 'Sujet',
            'dspan': (215, 220),
            'span': (66, 71)
          },
          {
            'id': 15,
            'text': '"',
            'dspan': (220, 221),
            'span': (71, 72)
          },
          {
            'id': 16,
            'text': '.',
            'dspan': (221, 222),
            'span': (72, 73)
          }
        ]
      }
    ]
```

The expanded tokens always have the indexes that are tuple objects, instead of integers as usual. In this example, the expanded token is the token with the index `(12, 13)`. The tuple indicates that this token is expanded into the syntactic words with the indexes ranging from `12` to `13`. The syntactic words are organized into a list stored in the field `'expanded'` of the original token. *Note that, part-of-speech, morphological tagging, dependency parsing, and lemmatization always work with the syntactic words instead of the original token in such case*. That's why we will only see additional features added to the syntactic words while the original token remains unchanged with only the information of its text form and spans. *As a last note, Named Entity Recognition (NER) module by contrast only works with the original tokens instead of the syntactic words, so we will not see the NER tags for the syntactic words*.

1.9 Part-of-speech, Morphological tagging and Dependency parsing

NOTE: *Quick examples* might be helpful for using this function.

In *trankit*, part-of-speech, morphological tagging, and dependency parsing are jointly performed. The module can work with either untokenized or pretokenized inputs, at both sentence and document level.

1.9.1 Document-level processing

Untokenized input

The sample code for this module is:

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
doc_text = '''Hello! This is Trankit.'''

all = p.posdep(doc_text)
```

Trankit first performs tokenization and sentence segmentation for the input document, then performs the part-of-speech, morphological tagging, and dependency parsing for the tokenized document. The output of the whole process is a native Python dictionary with list of sentences, each sentence contains a list of tokens with the predicted part-of-speech, the morphological feature, the index of the head token, and the corresponding dependency relation for each token. The output would look like this:

```
{
  'text': 'Hello! This is Trankit.',  # input string
  'sentences': [ # list of sentences
    {
      'id': 1, 'text': 'Hello!', 'dspan': (0, 6), 'tokens': [...]
    },
    {
      'id': 2,  # sentence index
      'text': 'This is Trankit.',  'dspan': (7, 23), # sentence span
      'tokens': [ # list of tokens
        {
          'id': 1, # token index
          'text': 'This',  # text form of the token
          'upos': 'PRON',  # UPOS tag of the token
          'xpos': 'DT',    # XPOS tag of the token
          'feats': 'Number=Sing|PronType=Dem', # morphological feature of the token
          'head': 3,  # index of the head token
          'deprel': 'nsubj', # dependency relation for the token
          'dspan': (7, 11), # document-level span of the token
          'span': (0, 4) # sentence-level span of the token
        },
        {'id': 2...},
        {'id': 3...},
        {'id': 4...}
      ]
    }
  ]
}
```

Pretokenized input

In some cases, we might already have a tokenized document and want to use this module. Here is how we can do it:

```
pretokenized_doc = [
  ['Hello', '!'],
  ['This', 'is', 'Trankit', '.']
]

tagged_doc = p.posdep(pretokenized_doc)
```

Pretokenized inputs are automatically recognized by *Trankit*. That's why we don't have to specify any additional tag when calling the function `.posdep()`. The output in this case will be the same as in the previous case except that now we don't have any span information.

1.9.2 Sentence-level processing

Sometimes we want to use this module for sentence inputs. To achieve that, we can simply set `is_sent=True` when we call the function `.posdep()`:

Untokenized input

```
sent_text = '''This is Trankit.'''

tagged_sent = p.posdep(sent_text, is_sent=True)
```

Pretokenized input

```
pretokenized_sent = ['This', 'is', 'Trankit', '.']

tagged_sent = p.posdep(pretokenized_sent, is_sent=True)
```

1.10 Lemmatization

NOTE: *Quick examples* might be helpful for using this function.

Trankit supports lemmatization for both untokenized and pretokenized inputs, at both sentence and document level. Here are some examples:

1.10.1 Document-level lemmatization

In this case, the input is assumed to be a document.

Untokenized input

```
from trankit import Pipeline

p = Pipeline('english')

doc_text = '''Hello! This is Trankit.'''

lemmatized_doc = p.lemmatize(doc_text)
```

Trankit would first perform tokenization and sentence segmentation for the input document. Next, it assigns a lemma to each token in the sentences. The output would look like this:

```
{
  'text': 'Hello! This is Trankit.',  # input string
  'sentences': [ # list of sentences
    {
      'id': 1, 'text': 'Hello!', 'dspan': (0, 6), 'tokens': [...]
    },
    {
      'id': 2,  # sentence index
      'text': 'This is Trankit.',  'dspan': (7, 23), # sentence span
      'tokens': [ # list of tokens
        {
          'id': 1, # token index
          'text': 'This',
          'lemma': 'this', # lemma of the token
          'dspan': (7, 11), # document-level span of the token
          'span': (0, 4) # sentence-level span of the token
        },
        {'id': 2...},
        {'id': 3...},
        {'id': 4...}
      ]
    }
  ]
}
```

For illustration purpose, we only show the first sentence.

Pretokenized input

Pretokenized inputs are automatically recognized by *Trankit*. The following snippet performs lemmatization on a pretokenized document, which is a list of lists of strings:

```
from trankit import Pipeline

p = Pipeline('english')

pretokenized_doc = [
  ['Hello', '!'],
  ['This', 'is', 'Trankit', '.']
]

lemmatized_doc = p.lemmatize(pretokenized_doc)
```

The output will look slightly different without the spans of the sentences and the tokens:

```
{
  'sentences': [
    {
      'id': 1,
      'tokens': [
        {
          'id': 1,
          'text': 'Hello',
          'lemma': 'hello'
        },
        {
          'id': 2,
          'text': '!',
          'lemma': '!'
        }
      ]
    },
    {
      'id': 2,
      'tokens': [
        {
          'id': 1,
          'text': 'This',
          'lemma': 'this'
        },
        {
          'id': 2,
          'text': 'is',
          'lemma': 'be'
        },
        {
          'id': 3,
          'text': 'Trankit',
          'lemma': 'trankit'
        },
        {
```

(continues on next page)

(continued from previous page)

```
          'id': 4,
          'text': '.',
          'lemma': '.'
        }
      ]
    }
  ]
}
```

1.10.2 Sentence-level lemmatization

Lemmatization module also accepts inputs as sentences. This can be done by setting the tag `is_sent=True`. The output would be a dictionary with a list of lemmatized tokens.

Untokenized input

```
from trankit import Pipeline

p = Pipeline('english')

sent_text = '''This is Trankit.'''

lemmatized_sent = p.lemmatize(sent_text, is_sent=True)
```

Pretokenized input

```
from trankit import Pipeline

p = Pipeline('english')

sent_text = '''This is Trankit.'''

pretokenized_sent = ['This', 'is', 'Trankit', '.']

lemmatized_sent = p.lemmatize(pretokenized_sent, is_sent=True)
```

1.11 Named entity recognition

NOTE: *Quick examples* might be helpful for using this function.

Currently, *Trankit* provides the Named Entity Recognition (NER) module for 8 languages which are Arabic, Chinese, Dutch, English, French, German, Russian, and Spanish. The NER module accepts the inputs that can be untokenized or pretokenized, at both sentence and document level. Below are some examples:

1.11.1 Document-level

Untokenized input

```
from trankit import Pipeline
# initialize a pipeline for English
p = Pipeline('english')

# a non-empty string to process, which can be a document or a paragraph with multiple␣
↪sentences
doc_text = '''Hello! This is Trankit.'''

tagged_doc = p.ner(doc_text)
```

The output would look like this:

```
{
  'text': 'Hello! This is Trankit.',  # input string
  'sentences': [ # list of sentences
    {
      'id': 1, 'text': 'Hello!', 'dspan': (0, 6), 'tokens': [...]
    },
    {
      'id': 2,  # sentence index
      'text': 'This is Trankit.',  'dspan': (7, 23), # sentence span
      'tokens': [ # list of tokens
        {
          'id': 1, # token index
          'text': 'This',
          'ner': 'O', # ner tag of the token
          'dspan': (7, 11), # document-level span of the token
          'span': (0, 4) # sentence-level span of the token
        },
        {'id': 2...},
        {'id': 3...},
        {'id': 4...}
      ]
    }
  ]
}
```

Pretokenized input

```
from trankit import Pipeline

p = Pipeline('english')

pretokenized_doc = [
  ['Hello', '!'],
  ['This', 'is', 'Trankit', '.']
]
```

(continues on next page)

(continued from previous page)

```
tagged_doc = p.ner(pretokenized_doc)
```

The output will look the same as in the untokenized case, except that now we don't have the text form for the input document as well as the span information for the sentences and the tokens.

1.11.2 Sentence-level

To enable the NER module to work with sentence-level instead of document-level inputs, we can set the tag `is_sent=True`:

Untokenized input

```
from trankit import Pipeline

p = Pipeline('english')

sent_text = 'This is Trankit.'

tagged_sent = p.ner(sent_text, is_sent=True)
```

Pretokenized input

```
from trankit import Pipeline

p = Pipeline('english')

pretokenized_sent = ['This', 'is', 'Trankit', '.']

tagged_sent = p.ner(pretokenized_sent, is_sent=True)
```

1.12 Building a customized pipeline

NOTE: Please check out the list of supported languages to check if the language of your interest is already supported by Trankit.

Building customized pipelines are easy with Trankit. The training is done via the class `TPipeline` and the loading is done via the class `Pipeline` as usual. To achieve this, customized pipelines are currently organized into 4 categories:

- `customized-mwt-ner`: a pipeline of this category consists of 5 models: (i) joint token and sentence splitter, (ii) multi-word token expander, (iii) joint model for part-of-speech tagging, morphologicial feature tagging, and dependency parsing, (iv) lemmatizer, and (v) named entity recognizer.
- `customized-mwt`: a pipeline of this category doesn't have the (v) named entity recognizer.
- `customized-ner`: a pipeline of this category doesn't have the (ii) multi-word token expander.
- `customized`: a pipeline of this cateogry doesn't have the (ii) multi-word token expander and the (v) named entity recognizer.

The category names are used as the special identities in the `Pipeline` class. Thus, we need to choose one of the categories to start training our customized pipelines. Below we show the example for training and loading a `customized-mwt-ner` pipeline. Training procedure for customized pipelines of other categories can be obtained by obmitting the steps related to the models that those pipelines don't have. For data format, `Tpipeline` accepts training data in CONLL-U format for the Universal Dependencies tasks, and training data in BIO format for the NER task.

1.12.1 Training

Training a joint token and sentence splitter

```
import trankit

# initialize a trainer for the task
trainer = trankit.TPipeline(
    training_config={
    'category': 'customized-mwt-ner', # pipeline category
    'task': 'tokenize', # task name
    'save_dir': './save_dir', # directory for saving trained model
    'train_txt_fpath': './train.txt', # raw text file
    'train_conllu_fpath': './train.conllu', # annotations file in CONLLU format for␣
↪training
    'dev_txt_fpath': './dev.txt', # raw text file
    'dev_conllu_fpath': './dev.conllu' # annotations file in CONLLU format for␣
↪development
    }
)

# start training
trainer.train()
```

Training a multi-word token expander

```
import trankit

# initialize a trainer for the task
trainer = trankit.TPipeline(
    training_config={
    'category': 'customized-mwt-ner', # pipeline category
    'task': 'mwt', # task name
    'save_dir': './save_dir', # directory for saving trained model
    'train_conllu_fpath': './train.conllu', # annotations file in CONLLU format  for␣
↪training
    'dev_conllu_fpath': './dev.conllu' # annotations file in CONLLU format for␣
↪development
    }
)

# start training
trainer.train()
```

Training a joint model for part-of-speech tagging, morphologicial feature tagging, and dependency parsing

```
import trankit

# initialize a trainer for the task
trainer = trankit.TPipeline(
    training_config={
    'category': 'customized-mwt-ner', # pipeline category
    'task': 'posdep', # task name
    'save_dir': './save_dir', # directory for saving trained model
    'train_conllu_fpath': './train.conllu', # annotations file in CONLLU format  for␣
↪training
    'dev_conllu_fpath': './dev.conllu' # annotations file in CONLLU format for␣
↪development
    }
)

# start training
trainer.train()
```

Training a lemmatizer

```
import trankit

# initialize a trainer for the task
trainer = trankit.TPipeline(
    training_config={
    'category': 'customized-mwt-ner',  # pipeline category
    'task': 'lemmatize', # task name
    'save_dir': './save_dir', # directory for saving trained model
    'train_conllu_fpath': './train.conllu', # annotations file in CONLLU format  for␣
↪training
    'dev_conllu_fpath': './dev.conllu' # annotations file in CONLLU format for␣
↪development
    }
)

# start training
trainer.train()
```

Training a named entity recognizer

Training data for a named entity recognizer should be in the BIO format in which the first column contains the words and the last column contains the BIO annotations. Users can refer to this file to get the sample data in the required format.

```
import trankit

# initialize a trainer for the task
```

(continues on next page)

(continued from previous page)

```
trainer = trankit.TPipeline(
    training_config={
    'category': 'customized-mwt-ner',  # pipeline category
    'task': 'ner', # task name
    'save_dir': './save_dir', # directory to save the trained model
    'train_bio_fpath': './train.bio', # training data in BIO format
    'dev_bio_fpath': './dev.bio' # training data in BIO format
    }
)

# start training
trainer.train()
```

A colab tutorial on how to train, evaluate, and use a custom NER model is also available here. Thanks @mrshu for contributing this to Trankit.

1.12.2 Loading

After the training steps, we need to verify that all the models of our customized pipeline are trained. The following code shows how to do it:

```
import trankit

trankit.verify_customized_pipeline(
    category='customized-mwt-ner', # pipeline category
    save_dir='./save_dir', # directory used for saving models in previous steps
    embedding_name='xlm-roberta-base' # embedding version that we use for training our␣
↪customized pipeline, by default, it is `xlm-roberta-base`
)
```

If the verification is success, this would printout the following:

```
Customized pipeline is ready to use!
It can be initialized as follows:

from trankit import Pipeline
p = Pipeline(lang='customized-mwt-ner', cache_dir='./save_dir')
```

From now on, the customized pipeline can be used as a normal pretrained pipeline.

The verification would fail if some of the expected model files of the pipeline are missing. This can be solved via the handy function `download_missing_files`, which is created for borrowing model files from pretrained pipelines provided by Trankit. Suppose that the language of your customized pipeline is English, the function can be used as below:

```
import trankit

trankit.download_missing_files(
        category='customized-ner',
        save_dir='./save_dir',
        embedding_name='xlm-roberta-base',
        language='english'
)
```

where `category` is the category that we specified for the customized pipeline, `save_dir` is the path to the directory that we saved the customized models, `embedding_name` is the embedding that we used for the customized pipeline (which is `xlm-roberta-base` by default if we did not specify this in the training process), and `language` is the language with the pretrained models that we want to borrow. For example, if we only trained a NER model for the customized pipeline, the snippet above would borrow the trained models for all the other pipeline components and print out the following message:

```
# Missing ./save_dir/xlm-roberta-base/customized-ner/customized-ner.tokenizer.mdl
# Missing ./save_dir/xlm-roberta-base/customized-ner/customized-ner.tagger.mdl
# Missing ./save_dir/xlm-roberta-base/customized-ner/customized-ner.vocabs.json
# Missing ./save_dir/xlm-roberta-base/customized-ner/customized-ner_lemmatizer.pt
# http://nlp.uoregon.edu/download/trankit/v1.0.0/xlm-roberta-base/english.zip
# Downloading: 100%|| 47.9M/47.9M [00:00<00:00, 89.2MiB/s]
# Copying ./save_dir/xlm-roberta-base/english/english.tokenizer.mdl to ./save_dir/xlm-
↪roberta-base/customized-ner/customized-ner.tokenizer.mdl
# Copying ./save_dir/xlm-roberta-base/english/english.tagger.mdl to ./save_dir/xlm-
↪roberta-base/customized-ner/customized-ner.tagger.mdl
# Copying ./save_dir/xlm-roberta-base/english/english.vocabs.json to ./save_dir/xlm-
↪roberta-base/customized-ner/customized-ner.vocabs.json
# Copying ./save_dir/xlm-roberta-base/english/english_lemmatizer.pt to ./save_dir/xlm-
↪roberta-base/customized-ner/customized-ner_lemmatizer.pt
```

After this, we can go back to do the verification step again.

1.13 Command-line interface

Starting from version v1.0.0, Trankit supports processing text via command-line interface. This helps users who are not familiar with Python programming language can use Trankit more easily.

1.13.1 Requirements

Users need to install Trankit via one of the following methods:

Pip:

```
pip install trankit==1.1.0
```

From source:

```
git clone https://github.com/nlp-uoregon/trankit
cd trankit
pip install -e .
```

1.13.2 Syntax

```
python -m trankit [OPTIONS] --embedding xlm-roberta-base --cpu --lang english --input␣
↪path/to/your/input --output_dir path/to/your/output_dir
```

What this command does are:

- Forcing Trankit to run on CPU (`--cpu`). Without this `--cpu`, Trankit will run on GPU if a GPU device is available.
- Initializing an English pipeline with XLM-Roberta base as the multilingual embedding (`--embedding xlm-roberta-base`).
- Performing all tasks on the input stored at `path/to/your/input` which can be a single input file or a folder storing multiple input files (`--input path/to/your/input`).
- Writing the output to `path/to/your/output_dir` which stores the output files, each is a json file with the prefix is the file name of the processed input file (`--output_dir path/to/your/output_dir`).

In this command, we can put more processing options at `[OPTIONS]`. Detailed description of the options that can be used:

- `--lang`

  Language(s) of the pipeline to be initialized. Check out this page to see the available language names.

  Example use:

  -Monolingual case:

  ```
  python -m trankit [other options] --lang english
  ```

  -Multilingual case with 3 languages:

  ```
  python -m trankit [other options] --lang english,chinese,arabic
  ```

  -Multilingual case with all supported languages:

  ```
  python -m trankit [other options] --lang auto
  ```

  In multilingual mode, trankit will automatically detect the language of the input file(s) to use corresponding models.

  Note that, language detection is done at file level.

- `--cpu`

  Forcing trankit to run on CPU. Default: False.Example use:

  ```
  python -m trankit [other options] --cpu
  ```

- `--embedding`

  Multilingual embedding for trankit. Default: xlm-roberta-base.

  Example use:

  -XLM-Roberta base:

  ```
  python -m trankit [other options] --embedding xlm-roberta-base
  ```

  -XLM-Roberta large:

```
python -m trankit [other options] --embedding xlm-roberta-large
```

- `--cache_dir`

  Location to store downloaded model files. Default: "cache/trankit".

  Example use:

```
python -m trankit [other options] --cache_dir your/cache/dir
```

- `--input`

  Location of the input.

  If it is a directory, trankit will process each file in the input directory at a time.

  If it is a file, trankit will process the file only.

  Example use:

  -Input is a directory:

```
python -m trankit [other options] --input some_dir_path
```

  -Input is a file:

```
python -m trankit [other options] --input some_file_path
```

- `--input_format`

  Indicating the input format.

  Case 1: Each input file is a single raw DOCUMENT string:

```
python -m trankit [other options] --input_format plaindoc
```

  Case 2: Each input file contains multiple raw SENTENCE strings in each line:

```
python -m trankit [other options] --input_format plainsen
```

  Case 3: Each input file contains pretokenized SENTENCES separated by "\n\n", each sentence is organized into multiple lines, each line contains only a single word:

```
python -m trankit [other options] --input_format pretok
```

  Sample inputs can be found here:

  plaindoc

  plainsen

  pretok

- `--output_dir`

  Location of the output directory to store the processed files. Processed files will be in json format, with the naming convention as follows:

```
processed_file_name = input_file_name + .processed.json
```

  Example use:

```
python -m trankit [other options] --output_dir some_dir_path
```

- `--task`

  Task to be performed for the provided input.

  Use cases:

  -Sentence segmentation, assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task ssplit
```

  -Sentence segmentation + Tokenization, assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task dtokenize
```

  -Tokenization only, assuming each input file contains multiple raw SENTENCE strings in each line (`--input_format plainsen`).

```
python -m trankit [other options] --task stokenize
```

  -Sentence segmentation, Tokenization, Part-of-speech tagging, Morphological tagging, Dependency parsing.

  Assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task dposdep
```

  -Tokenization only, Part-of-speech tagging, Morphological tagging, Dependency parsing.

  Assuming each input file contains multiple raw SENTENCE strings in each line (`--input_format plainsen`).

```
python -m trankit [other options] --task sposdep
```

  -Part-of-speech tagging, Morphological tagging, Dependency parsing.

  Assuming each input file contains pretokenized SENTENCES separated by "\n\n", each sentence is organized into multiple lines, each line contains only a single word (`--input_format pretok`).

```
python -m trankit [other options] --task pposdep
```

  -Sentence segmentation, Tokenization, Lemmatization

  Assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task dlemmatize
```

  -Tokenization only, Lemmatization

  Assuming each input file contains multiple raw SENTENCE strings in each line (`--input_format plainsen`).

```
python -m trankit [other options] --task slemmatize
```

  -Lemmatization

  Assuming each input file contains pretokenized SENTENCES separated by "\n\n", each sentence is organized into multiple lines, each line contains only a single word (`--input_format pretok`).

```
python -m trankit [other options] --task plemmatize
```

-Sentence segmentation, Tokenization, Named Entity Recognition.

Assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task dner
```

-Tokenization only, Named Entity Recognition.

Assuming each input file contains multiple raw SENTENCE strings in each line (`--input_format plainsen`).

```
python -m trankit [other options] --task sner
```

-Named Entity Recognition.

Assuming each input file contains pretokenized SENTENCES separated by "\n\n", each sentence is organized into multiple lines, each line contains only a single word (`--input_format pretok`).

```
python -m trankit [other options] --task pner
```

-Sentence segmentation, Tokenization, Part-of-speech tagging, Morphological tagging, Dependency parsing, Named Entity Recognition.

Assuming each input file is a single raw DOCUMENT string (`--input_format plaindoc`).

```
python -m trankit [other options] --task dall
```

-Tokenization only, Part-of-speech tagging, Morphological tagging, Dependency parsing, Named Entity Recognition.

Assuming each input file contains multiple raw SENTENCE strings in each line (`--input_format plainsen`).

```
python -m trankit [other options] --task sall
```

-Part-of-speech tagging, Morphological tagging, Dependency parsing, Named Entity Recognition.

Assuming each input file contains pretokenized SENTENCES separated by "\n\n", each sentence is organized into multiple lines, each line contains only a single word (`--input_format pretok`).

```
python -m trankit [other options] --task pall
```