Visualizing Vietnamese Dependency Parsing Result

A web application to demo Dependency Parsing models

Toan T. Le (toan.le2019@ict.jvn.edu.vn)

Thuy-Trang .T Le (trang.le2019@qcf.jvn.edu.vn)

John von Neumann Institute, Ho Chi Minh National University, Vietnam

July 2021

Contents

Section 1

Introduction

Dependency Parsing is one of the fundamental tasks in Natural Language Processing (NLP). It focuses on extracting syntactic structure through the words of a sentence and a set of directed binary relations among those words. The parse result provides extremely useful features for other downstream tasks such as machine translation, information extraction, question answering, co-reference resolution, etc.

ID	Form	POS	Head	DepRel
1	Tôi $_{I}$	PRON	2	sub
2	là $_{am}$	VERB	0	root
3	sinh_viên $_{student}$	NOUN	2	vmod

Figure 1.1: Example of Dependency Parsing result

Dependency Parsing research for Vietnamese language has been explored in the past 8 years. It started with the release of Vietnamese Dependency Treebank (VnDT) [5]. Since then, dependency parsing models for Vietnamese language has been improved over the years, Transition-based with Selectional Branching model [1] was applied to Vietnamese through VNCoreNLP toolkit [10], Joint model for POS Tagging and Dependency Parsing using BiLSTM and BIST graph-based parser [7], and the current State-of-the-Art model is PhoNLP [8] which use FFNN and Biaffine Classifier [2] on PhoBERT word encoding [4].

This project aims to investigate Vietnamese Dependency Parsing models, the demo them using a web application.

Our contributions:

- Investigate and incorporate three Vietnamese Dependency Parsing models into a com-

plete web application project.

- Re-structure and expand the jPTDP project so that it can be installed as a package via pip install, as well as allowing jPTDP model to be called directly from Python code (not just from command line originally).
- Train the jPTDP model, and make it available to use immediately through downloading the pre-trained model's files.
- Build a Single Page Application, using Django and VueJs, to demo all the parsing models.
- Design an interface to show the Dependency Parsing result, via Table and Tree Graph, for easy visualization.

The project is uploaded publicly on Github `https://github.com/lethien/dependency_parsing_webapp`

Section 2

Dependency Parsing

2.1 Formalism [3]

Dependency Parsing targets the Dependency Structure of a sentence.

Dependency Structure can be formalized as a directed graph G = (V, A), in which:

- V is the set of vertices, which contains the set of words in the given sentence.
- A is the set of binary directed arcs, which corresponding to the head-dependent and grammatical function relationships between pairs of words in V.

Considering further constraints, the directed graph G specifics to a dependency tree that satisfies the following:

- There is single root node that has no incoming arc.
- Except for the root node, each vertex has exactly one incoming arc.
- There is a unique path from the root node to each vertex in V.

Each head→dependent relationship between the words can be labeled or not. If all relationships contain grammatical function information, the Dependency Tree is called labeled tree. The list of labels is defined in Universal Dependencies project [9].
Note that there is an optional constrain on dependency tree, which is Projectivity. An arc is considered projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is considered projective if all of it's arcs are projective. This constrain depends on the words order in the sentence. For languages that has flexible word order such as Vietnamese, the dependency tree can be non-projective, which will lead to errors in Transition-Based methods.

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Figure 2.1: Examples of Dependency labels from Universal Dependencies project

2.2 Dependency Treebank

Dependency Treebank is the data used in developing, training, and evaluating dependency parser. Treebanks for multiple languages can be found on Universal Dependencies project [9] website.

Dependency Treebanks follow the CoNLL-U Format which comprises 10 columns:

- ID: Word index, integer starting at 1.
- FORM: Word form or punctuation symbol.
- LEMMA: Lemma or stem of word form.
- UPOS: Universal part-of-speech tag.
- XPOS: Language-specific part-of-speech tag.
- FEATS: List of morphological features.
- HEAD: Head of the current word.
- DEPREL: Universal dependency relation to the HEAD.
- DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
- MISC: Any other annotation.

For Vietnamese language, and for this project, we typically use the Vietnamese Dependency Treebank (VnDT) [5].

2.3 VNCoreNLP

VNCoreNLP is a Vietnamese natural language processing toolkit. VNCoreNLP helps provide a rich set of linguistic annotations, including word segmentation, pos tagging, named entity recognition, and dependency parsing.

Among the components of VNCoreNLP, word segmentation, specifically RDRsegmenter [6], is extremely useful as the key first pre-processing step when dealing with Vietnamese Language. In English, words can be separated by white spaces. But in Vietnamese, white spaces can also be used to separate syllables of words. We will use this word segmentation function as the pre-processing step before feeding the sentences into all Dependency Parsing models.

As for dependency parsing, VNCoreNLP expands the Transition-based with Selectional Branching model [1] with greedy approach.

To use VNCoreNLP in your project, Python 3.4+ and Java 1.8+ are required.

VNCoreNLP can be installed using pip:

```
# pip install vncorenlp
```

Basic syntax:

```
from vncorenlp import VnCoreNLP

annotator = VnCoreNLP("<FULL-PATH-to-VnCoreNLP-jar-file>",
                      annotators="wseg,pos,ner,parse",
                      max_heap_size='-Xmx2g')

text = "Vietnamese sentences separated by ."

annotated_text = annotator.annotate(text)
```

The result of annotated text:

```
{'sentences': [
         [{'index': 1,
           'form': 'Ông',
           'posTag': 'Nc',
           'nerLabel': 'O',
           'head': 4,
```

```
            'depLabel': 'sub'
            },
            ...other words...
           ],
           ...other sentences...
]}
```

We can not find the code to re-train the dependency parsing model of VNCoreNLP. We tried using other implementation of Transition-Based methods, but all end up failing when facing with the Non-Projectivity problem stated previously. So for this project, we will use the pre-trained model of VNCoreNLP.

2.4 Joint model for POS Tagging and Dependency Parsing - jPTDP [7]

jPTDP is a neural network based model that jointly learn POS tagging and Dependency Parsing. The structure of jPTDP is comprised of 3 parts:

- Word vector representation
- POS tagging component
- Dependency Parsing component

Word vector representation transform each word in the input sentence into two vectors: word embedding (100-dimension, can be changed if re-train) and character embedding (50-dimension, can be changed if re-train). Both embeddings are initialized randomly, then trained via one-layer BiLSTM.

POS tagging component takes the word embedding as input, and feeds it through a BiLSTM to get pos embedding. The pos embedding will be further fed into a MLP with softmax output to get the POS tag.

Dependency Parsing component incorporates all three word embedding and character embedding and pos embedding into a sequence vector, then feeds it through a BiLSTM to get a latent feature vector. Then latent feature vectors of each pair of words are fed into:

- Arc scoring, which is an MLP with one-node output, to score each possible arc. The score is high for correct arcs, and low otherwise. The result will be the dependency tree that maximizes the total scores of it's arc.

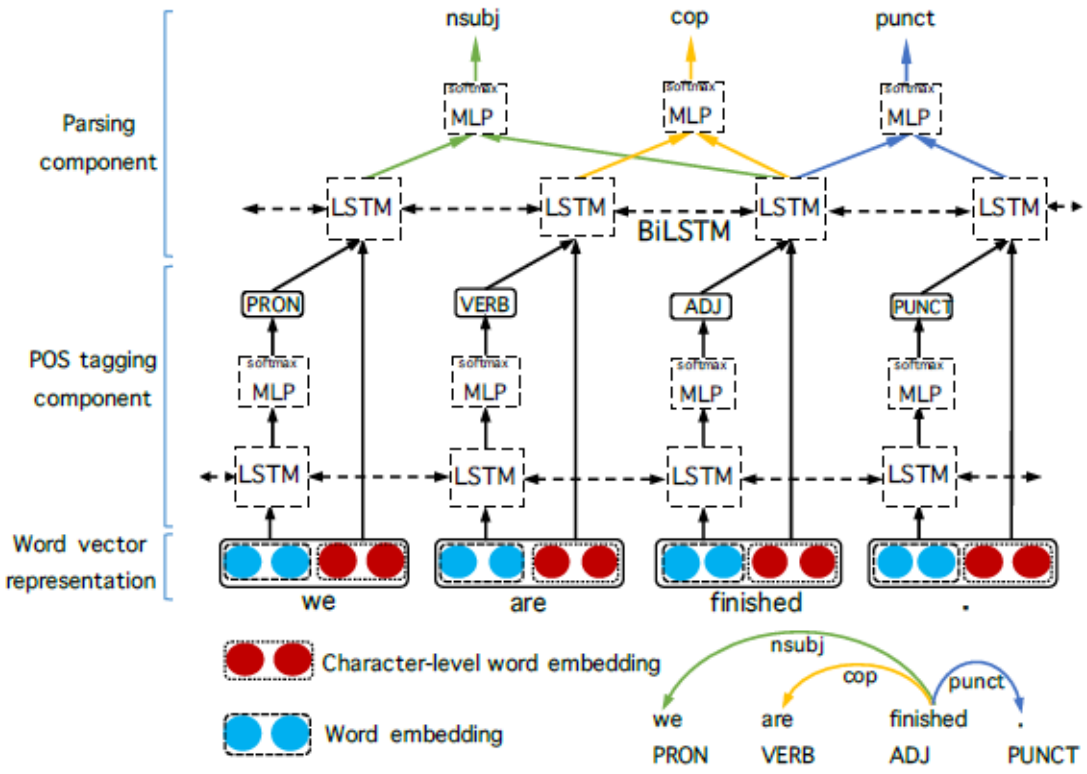


Figure 2.2: Structure of jPTDP model

- Arc labeling, which is an MLP with soft-max output, to predict the label of each arc in the built dependency tree.

jPTDP has been added to this project to use. The original jPTDP project only support invoking through command line. Our contributions are:

- Re-structure the project so that it can be called directly in python code.
- Re-structure the project so that it can be installed as a package via pip install.
- Re-train the model using provided VnDT dataset. The result model includes two files, model and model.params. They are uploaded to Google Drive and will be download to corrected folder when setting up the project with running the setup_models.py file (More information will be provided in the setup section).

jPTDP can not be installed using pip:

```
# cd jPTDP
# pip install .
# cd ..
```

Basic syntax:

```
from jPTDP.jPTDP import jPTDP_model
from vncorenlp import VnCoreNLP

vncorenlp_model = VnCoreNLP("<FULL-PATH-to-VnCoreNLP-jar-file>",
                      annotators="wseg,pos,ner,parse",
                      max_heap_size='-Xmx2g')

jptdp_model = jPTDP_model()

text = "Vietnamese sentences separated by ."

sentences = vncorenlp_model.tokenize(text)
sentences = [' '.join(s) for s in sentences]

parse_result = jptdp_model.annotate(sentences)
```

The result of parse result:

```
[
    [
        ConllEntryObject(id, form, pred_pos, pred_parent_id, pred_relation),
        ...other words...
    ],
    ...other sentences...
]
```

If you want to re-train the jPTDP model with your data, run the following:

```
# cd jPTDP
# python jPTDPScript.py
    --dynet-seed 123456789 --dynet-mem 1000 --epochs 30
    --lstmdims 128 --lstmlayers 2 --hidden 100
    --wembedding 100 --cembedding 50 --pembedding 100
    --outdir jPTDP/model/
    --train [PATH_TO_CoNLLU_TRAIN_DATA]
    --dev [PATH_TO_CoNLLU_DEV_DATA]
```

2.5 PhoNLP [8]

PhoNLP is the current SOTA model of Vietnamese Dependency Parsing task.
Comparing to the jPTDP model:

- PhoNLP jointly learn three tasks, including POS tagging, named entity recognition, and Dependency Parsing.
- PhoNLP use contextualized latent feature embeddings from PhoBERT [4] as the input to deep learning layers.

The structure of PhoNLP is comprised of 4 parts:

- BERT-based encoder, which is the output of PhoBERT
- POS tagging component
- Named entity recognition (NER) component
- Dependency Parsing component

The POS tagging component takes the feature embeddings from PhoBERT, and feeds it into a Feed-Forward network (FFNN) with a softmax output to get the prediction for all POS labels. The POS result is the label with maximum score.

The NER component takes both the feature embeddings and POS embeddings, and feeds them through a FFNN, and then through a Conditional Random Field (CRF) for NER label.

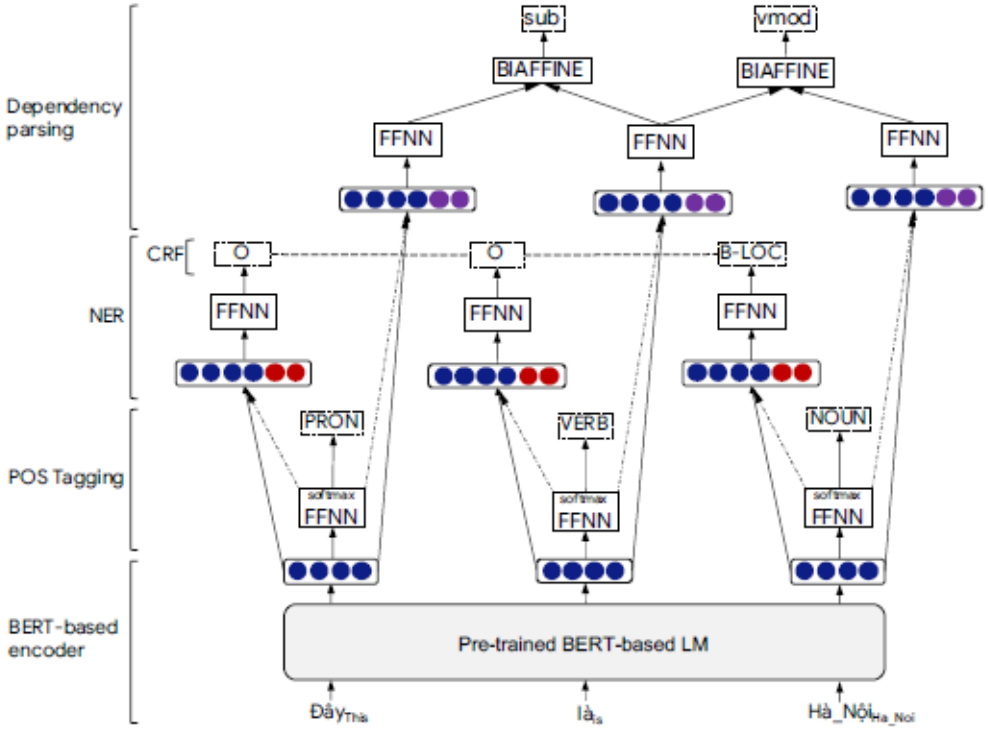
The Dependency Parsing component takes both the feature embeddings and POS embeddings, then feeds into an FFNN to get head-dependent representations. Then the representations will be fed into:

- Arc scoring, which is an Biaffine classifier [2], to score each possible arc. The score is high for correct arcs, and low otherwise. The result will be the dependency tree that maximizes the total scores of it's arc.
- Arc labeling, which is an Biaffine classifier [2], to predict the label of each arc in the built dependency tree.

To use PhoNLP in your project, Python 3.6+ and Pytorch 1.4+ are required.

PhoNLP can be installed using pip:

```
# pip install phonlp
```



ID	Form	POS	NER	Head	DepRel
1	Đây$_{This}$	PRON	O	2	sub
2	là$_{is}$	VERB	O	0	root
3	Hà_Nội$_{Ha_Noi}$	NOUN	B-LOC	2	vmod

Figure 2.3: Structure of PhoNLP model

Basic syntax:

```
import phonlp
from vncorenlp import VnCoreNLP

vncorenlp_model = VnCoreNLP("<FULL-PATH-to-VnCoreNLP-jar-file>",
                        annotators="wseg,pos,ner,parse",
                        max_heap_size='-Xmx2g')

text = "Vietnamese sentences separated by ."

sentences = vncorenlp_model.tokenize(text)

for sentence in sentences:
    parse_result = phonlp_model.annotate(text=' '.join(sentence))
```

The result of parse result of each sentence:

```
[
    [[wf1, wf2, ...other words' form...]],
    [[[pos1], [pos2], ...other words' pos taggings...]],
    [[ner1, ner2, ...other words' ner label...]],
    [[[h1-l1], [h2-l2], ...other words' relation (head-label)...]]
]
```

PhoNLP model, as well as the needed PhoBERT model, are SOTAs at the moment, and they take lots of time and computational resource to train. In the scope of this project, as well as our limitations, we decided to not re-train, but use the pre-trained PhoNLP model that is provided by the author.

If you want to re-train the PhoNLP model with your data, run the following:

```
# git clone https://github.com/VinAIResearch/PhoNLP
# cd PhoNLP
# pip install -e .
# cd phonlp/models
# python run_phonlp.py --mode train
    --save_dir <model_folder_path>
    --pretrained_lm <transformers_pretrained_model>
    --lr <float_value> --batch_size <int_value>
    --num_epoch <int_value> --lambda_pos <float_value>
    --lambda_ner <float_value> --lambda_dep <float_value>
    --train_file_pos <path_to_training_file_pos>
    --eval_file_pos <path_to_validation_file_pos>
    --train_file_ner <path_to_training_file_ner>
    --eval_file_ner <path_to_validation_file_ner>
    --train_file_dep <path_to_training_file_dep>
    --eval_file_dep <path_to_validation_file_dep>
```

Section 3

The Web Application

3.1 Main Components

There are three main components in the project:

- The Dependency Parsing models
- The VueJs Frontend
- The Django Backend

1. The Dependency Parsing models are integrated into the project through different ways:
   - The VNCoreNLP toolkit is installed as a package via pip install. And the pre-trained model's files are stored in webapp/dependency_api/vncorenlp/
   - The jPTDP model is stored in jPTDP folder. And the pre-trained model's files are stored in jPTDP/model/. Then the whole package is installed with pip install .
   - The PhoNLP model is installed as a package via pip install. And the pre-trained model's files are stored in webapp/dependency_api/pretrained_phonlp/
   - All pre-trained models' files can be downloaded into their respective location with the help of the Python script in setup_models.py
2. The VueJs Frontend code is located at webapp/frontend_code folder:
   - The package.json file contains all npm packages needed for the Frontend. All can be installed via npm install.
   - The vue.config.js file instructs all the built static resources will be stored in the static/src/vue/dist/ folder. This is where the Django Backend will get the static files needed for the Interface.

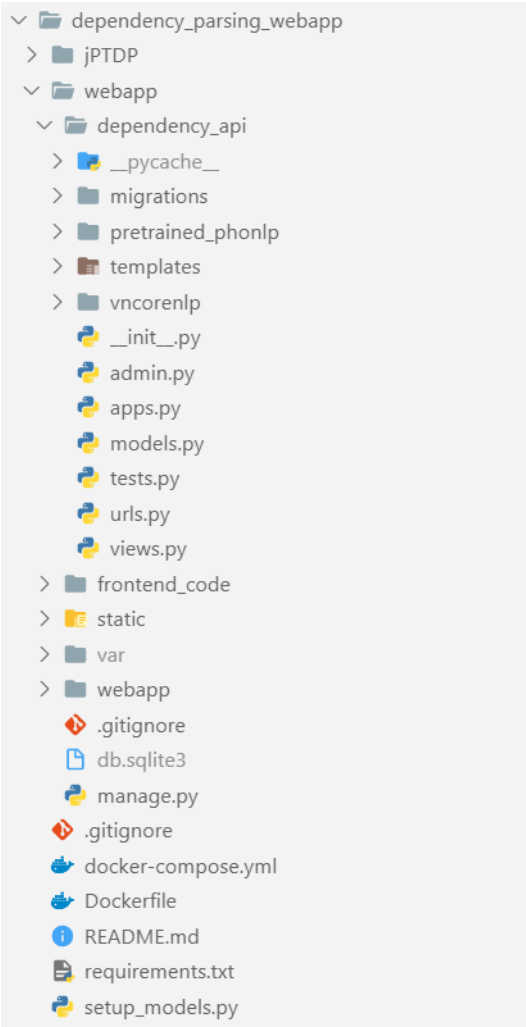


Figure 3.1: The Project Code structure

- The App.vue file is the entry point of this VueJs application.
- The components folder stores all vue files that define Vue components used in App.vue.

3. The Django Backend code is located at webapp folder:
   - The webapp/settings.py file instructs Django to collect all static files in the static/src/vue/dist/ folder (result of building VueJs Frontend). Django will then store those static files in the var/static_root folder (this is where Django get all the public files to serve the users).
   - dependenct_api is the main and only app in Django Backend. The urls.py and views.py files contains all the logic to handle user's requests, as well as using Parsing models to return Dependency Trees.
   - The dependency_api/templates/vue_templates/home_page.html file only contains a empty div with id="app" to be the place-holder for Vue to populate the contents. VueJs Fron-

tend built files are included via ¡script¿ tags. This is how this project incorporate VueJs into Django.

3.2 Setup Howto

This project requires:

- Python 3.6+
- Java 1.8+
- NodeJs Package Manager npm
- Yarn Package Manager yarn

Setup steps using command line:

1. Navigate the pointer to the root folder of the project. For example:

   ```
   cd project\_repos/dependency_parsing_webapp
   ```

2. Install python packages:

   ```
   pip install -r requirements.txt
   ```

3. Download and setup files for models:

   ```
   python setup_models.py
   ```

4. Install jPTDP:

   ```
   cd jPTDP
   pip install .
   cd ..
   ```

5. Setup VueJs and build static files:

   ```
   cd webapp/frontend_code
   npm install
   yarn serve
   (wait for server to start then exit with Ctrl-C)
   cd ../..
   ```

6. Setup Django and collect static files:

```
cd webapp
python manage.py collectstatic
cd ..
```

7. Start the web application:

```
cd webapp
python manage.py runserver
```

8. Open a web browser and access the web application by URL: http://[machine-ip]:8000

Section 4

Final Result

4.1 The Initial Homepage

The input form contains three main components:

- The input textarea, which allow user to put in Vietnamese sentences.
- The Model to use radio group, which allow user to choose which parsing model will handle the job.
- The Parse button, which submit the form via Ajax.

Figure 4.1: The Initial Home screen

4.2 The Loading Screen

When the user inputs the sentences, then choose the model, and finally click Parse:

- The whole form is disabled.
- The loading icon appears until the result is returned from the Backend.

Figure 4.2: The Loading screen

4.3 The Result Section

After receiving the result from Backend, the Result Section will appear below the form:

4.3.1 The Result Navigator

The Navigator allow user choose which sentence's result to show:

- Click Previous button to show the previous sentence
- Click Next button to show the next
- Previous button will be disabled if this is the first sentence
- Next button will be disabled if this is the last sentence

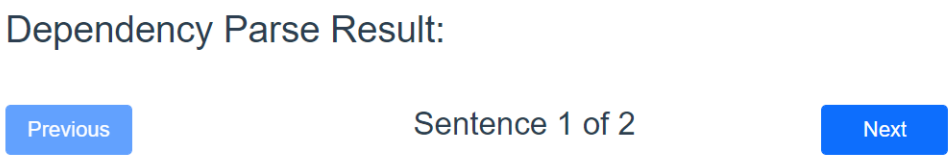


Figure 4.3: The Result Navigator

4.3.2 The Result Table

Based on the choice of user when using the Result Navigator, the chosen sentence's result will be shown in table form:

- ID column contains the id of each word in the sentence
- Form column contains the word itself
- POS column contains the POS tagging of each word in the sentence
- Head column contains the head's id of each word.
- Dependency Label column contains the label of the dependency relationship of each word in the sentence and their corresponding head.

Dependency Parse Table:

ID	Form	POS	Head	Dependency Label
1	Tôi	P	2	sub
2	là	V	0	root
3	sinh_viên	N	2	dob
4	cao_học	N	3	nmod
5	tại	E	3	loc
6	viện	N	5	pob
7	JVN	Ny	6	nmod
8	.	CH	2	punct

Figure 4.4: The Result Table

4.3.3 The Result Tree Graph

Based on the choice of user when using the Result Navigator, the chosen sentence's result will be shown in graph form:

- The Dependency Tree is drawn in a SVG tag
- If the tree is too deep or wide, some of the nodes can be hidden. User can drag (click and hold and move the cursor) the Tree Graph to move it around to check all the nodes.

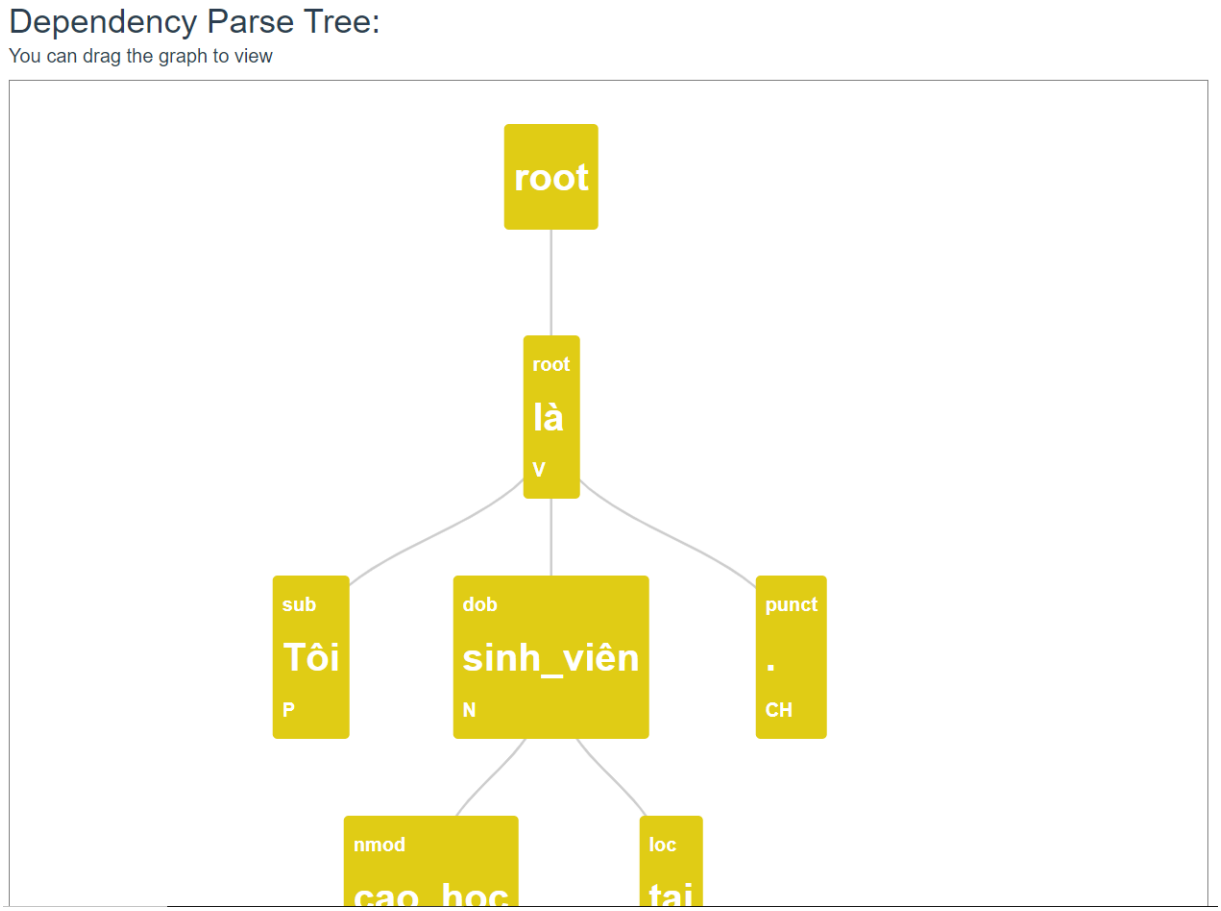


Figure 4.5: The Result Graph for Dependency Tree

Bibliography

[1] Choi, J. D. and McCallum, A. (2013). Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria. Association for Computational Linguistics.

[2] Dozat, T. and Manning, C. D. (2016). Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

[3] Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st edition.

[4] Nguyen, D. Q. and Nguyen, A. T. (2020). Phobert: Pre-trained language models for vietnamese. *CoRR*, abs/2003.00744.

[5] Nguyen, D. Q., Nguyen, D. Q., Pham, S. B., Nguyen, P.-T., and Nguyen, M. L. (2014). From Treebank Conversion to Automatic Dependency Parsing for Vietnamese. In *Proceedings of 19th International Conference on Application of Natural Language to Information Systems*, pages 196–207.

[6] Nguyen, D. Q., Nguyen, D. Q., Vu, T., Dras, M., and Johnson, M. (2018). A Fast and Accurate Vietnamese Word Segmenter. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*, pages 2582–2587.

[7] Nguyen, D. Q. and Verspoor, K. (2018). An improved neural network model for joint POS tagging and dependency parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 81–91, Brussels, Belgium. Association for Computational Linguistics.

[8] Nguyen, L. T. and Nguyen, D. Q. (2021). Phonlp: A joint multi-task learning model for vietnamese part-of-speech tagging, named entity recognition and dependency parsing. *CoRR*, abs/2101.01476.

[9] Nivre, J., de Marneffe, M., Ginter, F., Hajic, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F. M., and Zeman, D. (2020). Universal dependencies v2: An evergrowing multilingual treebank collection. *CoRR*, abs/2004.10643.

[10] Vu, T., Nguyen, D. Q., Nguyen, D. Q., Dras, M., and Johnson, M. (2018). VnCoreNLP: A Vietnamese natural language processing toolkit. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 56–60, New Orleans, Louisiana. Association for Computational Linguistics.