BENTO: A Visual Platform for Building Clinical NLP Pipelines Based on CodaLab

Yonghao Jin, Fei Li and **Hong Yu**
Department of Computer Science, University of Massachusetts
Lowell, MA, USA

Abstract

CodaLab[1] is an open-source web-based platform for collaborative computational research. Although CodaLab has gained popularity in the research community, its interface has limited support for creating reusable tools that can be easily applied to new datasets and composed into pipelines. In clinical domain, natural language processing (NLP) on medical notes generally involves multiple steps, like tokenization, named entity recognition, etc. Since these steps require different tools which are usually scattered in different publications, it is not easy for researchers to use them to process their own datasets. In this paper, we present *BENTO*, a workflow management platform with a graphic user interface (GUI) that is built on top of CodaLab, to facilitate the process of building clinical NLP pipelines. BENTO comes with a number of clinical NLP tools that have been pre-trained using medical notes and expert annotations and can be readily used for various clinical NLP tasks. It also allows researchers and developers to create their custom tools (e.g., pre-trained NLP models) and use them in a controlled and reproducible way. In addition, the GUI interface enables researchers with limited computer background to compose tools into NLP pipelines and then apply the pipelines on their own datasets in a "what you see is what you get" (WYSIWYG) way. Although BENTO is designed for clinical NLP applications, the underlying architecture is flexible to be tailored to any other domains.

1 Introduction

With the machine learning research going deep, computational models are becoming increasingly large with intensive hyper-parameters tuning, making the research difficult to reproduce. To tackle this problem, researchers have developed CodaLab as an open-source platform for researchers and software developers. However, CodaLab has limited support for reusable tools that can be easily applied to different datasets and be composed into computational pipelines.

[1] codalab.org

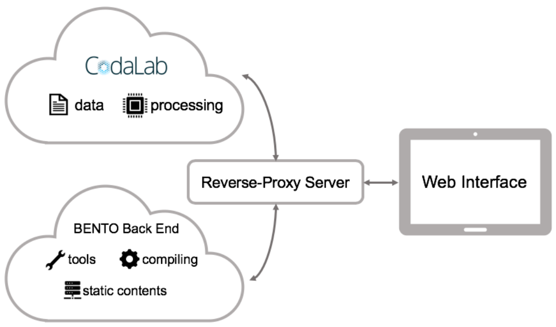


Figure 1: The architecture of BENTO. The BENTO back end stores the description files of various tools (e.g., pre-trained NLP models), processes static contents of the application and handles compilation of the user-defined pipelines. The CodaLab back end stores the datasets (bundles) and executes computational jobs. The two back end servers are brought behind a single domain name using a reverse proxy server.

Building pipelines is essential for the research of certain domains. Take the medical informatics research as an example, a complete NLP analysis on medical notes often involves multiple steps like tokenization, de-identification (Dernoncourt et al., 2017; Liu et al., 2017), entity recognition (Li et al., 2018; Xu et al., 2017; Jagannatha and Yu, 2016) and normalization (Li et al., 2019, 2017; Cho et al., 2017), relation extraction (Li et al., 2018; He et al., 2019), etc. Since these steps require different tools and these tools are usually scattered in different publications, it is far from trivial to leverage these tools on new datasets even though the authors have released the source code. Therefore, we developed a user-friendly workflow management platform, BiomEdical Nlp TOolkits (*BENTO*), to facilitate the process of building and applying of clinical NLP pipelines.

The architecture of BENTO is illustrated in Figure 1. BENTO has three main components. The web interface is supported by two back ends brought together by a reverse-proxy server. The CodaLab back end stores the datasets and executes computational jobs. The BENTO back end serves tool information and transforms user-defined pipelines to CodaLab commands.

The advantages of such architecture are two-fold. First, it is flexible to use CodaLab as the back end for adding custom tools (e.g., pre-trained NLP models) and processing data in a controlled and reproducible way. All the tools are containerized with Docker[2], which makes the platform to keep a unified interface to manage the models and not need to maintain different operating environment for different models. Second, the web interface makes it easier for users to construct NLP pipelines through editing flowcharts and then apply the pipelines to their data. The web-based architecture also makes the platform widely accessible without complex installation and configuration.

In this paper, we also show the examples of using BENTO to integrate several clinical NLP applications such as hypoglycemia detection (Jin et al., 2019) and adverse drug event extraction (Li et al., 2018), and build pipelines based on these tools. BENTO helps build NLP pipelines, which is a promising system to accelerate the medical informatics research.

2 Related Work

Galaxy (Afgan et al., 2018) is a similar computational platform that is focused in bioinformatics and computational biology, whose interface inspires the design of ours. The main restriction of the Galaxy platform is that users can only access the tools managed by administrators and cannot define their own tools. In linguistic research community, other related platforms include lingvis.io (El-Assady et al., 2019), which is focused on integrating NLP operations with visualizations , and Argo (Rak et al., 2012), a web-based text mining workbench based on the UIMA framework. Stanford CoreNLP (Manning et al., 2014) provides a commonly used NLP tool set. On the library level, NLTK (Hardeniya et al., 2016) is a popular Python library that integrates multiple widely used NLP tools. OpenNLP (Morton et al., 2005) is a Java library that provides machine learning based toolkits for NLP tasks. FudanNLP (Qiu et al., 2013) is a Java based library which integrates the machine learning models and datasets for Chinese NLP.

In the medical domain, NILE (Yu and Cai, 2013) is a Java package which includes rule based NLP methods for information extraction from medical notes. Apache cTAKES (Apache cTAKES, 2018) and CLAMP (Soysal et al., 2018) are two clinical NLP systems with pipeline-based architecture in the UIMA framework. Both systems have a graphical user interface, allowing users to build pipelines from build-in UIMA components. However, the UIMA framework has a steep learning curve. It is also not widely used in the machine-learning-based NLP research. Furthermore, most NLP applications are often released as command line programs. Therefore, it is hard to extend applications that use the UIMA framework with new models. In contrast, tools on our BENTO platform are based on command line programs and users can easily define their own tools with little restriction.

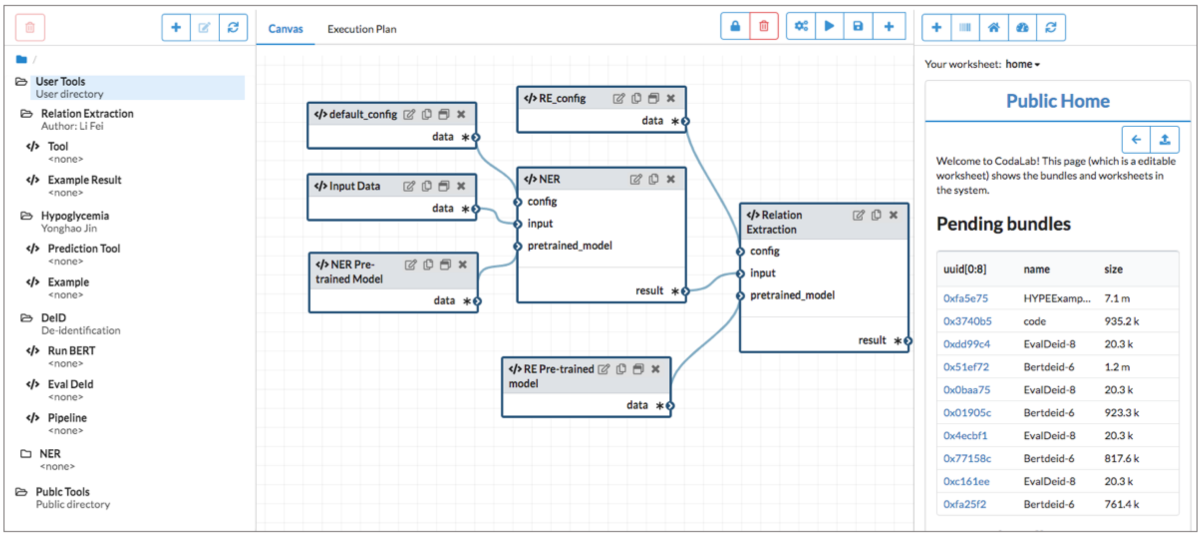
3 System Description

BENTO mainly comprises three parts: a front-end web application, a BENTO back end server and a CodaLab back end. As shown in Figure 1, BENTO has a web-based user interface, from which users can upload data, edit tools, submit jobs and perform various other operations. The BENTO back end is a web server that is mainly used for storing the tools, including the user-defined ones, so they can be accessed in different sessions. The CodaLab back end is used for execution of each computational job. When a tool is being executed, BENTO will generate a series of CodaLab commands based on the tool information and the input bundles. The outputs of the tool are the run bundles generated from those commands which can be passed on to the down-stream tools and inspected by the users on the CodaLab interface.

3.1 Web Interface

As shown in Figure 2, the user interface of our platform is a web application that can be roughly divided into three panels from left to right: tool panel, canvas panel and worksheet panel. The tool panel lists the current available tools on the platform organized in a hierarchical file system struc-

[2]docker.com



Your worksheet: home

Public Home

Welcome to CodaLab! This page (which is a editable worksheet) shows the bundles and worksheets in the system.

Pending bundles

uuid[0:8]	name	size
0xfa5e75	HYPEExamp...	7.1 m
0x3740b5	code	935.2 k
0xdd99c4	EvalDeid-8	20.3 k
0x51ef72	Bertdeid-6	1.2 m
0x0baa75	EvalDeid-8	20.3 k
0x01905c	Bertdeid-6	923.3 k
0x4ecbf1	EvalDeid-8	20.3 k
0x77158c	Bertdeid-6	817.6 k
0xc161ee	EvalDeid-8	20.3 k
0xfa25f2	Bertdeid-6	761.4 k

Figure 2: BENTO Web Interface. The interface can be roughly divided into three parts from left to right: tool panel, canvas panel and worksheet panel. The tool panel lists the current available tools organized in a tree view. The canvas panel contains the flowchart of the current pipeline. Every node represents a tool or dataset and each connection indicates the data flow in the pipeline.[3] This figure shows an example of the pipeline for entity and relation extraction. The worksheet panel displays the content of the CodaLab worksheet such as bundles and their UUIDs.

ture along with the meta information. Users can edit the *User Tools* folder using the buttons listed on the top menu bar. To run a tool, users can simply drag it to the canvas panel to the right and a tool node will appear on the canvas. A node, shown in the workflow Figure 2, contains several input and output ports, corresponding to the inputs and outputs of the tool.

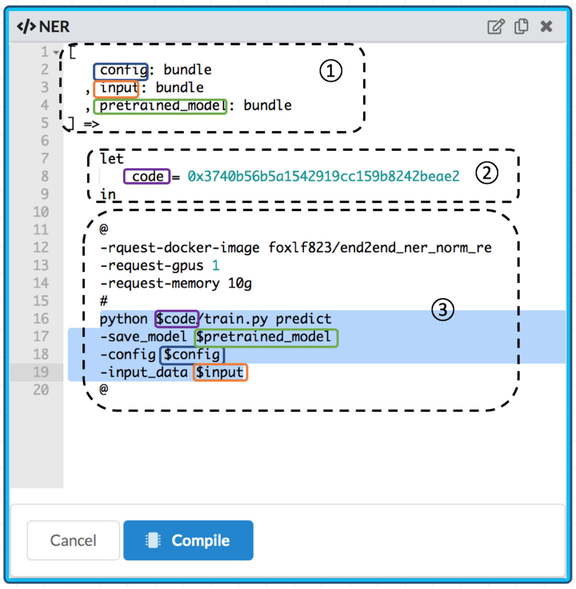
Tool nodes can be linked together to form a pipeline and the connections represent the data flow during execution (Figure 2). A connection starts from an output port and ends in an input port. An input port accepts only a single connection while an output port can initiate one or multiple connections. Users can edit the tool by clicking the *Editor button* (✎) on the top right corner and the node will be toggled to an editor interface (Figure 3). The editor contains the expression of the tool (Section 3.3), which can be modified by the users. The rightmost part is the worksheet panel that displays the content of the current selected worksheet. Worksheets are editable markup documents provided by CodaLab. Dragging a bundle entry from the worksheet panel to the canvas will create a data node. A data node is similar to the tool node except that it does not have any input port which naturally represents a data entity in a computational pipeline.

[3]For simplicity, pre-processing steps like tokenization is built-in in each tool.

3.2 CodaLab Back End

An important design goal of BENTO is flexibility. Users should be able to easily define their own tools on the BENTO platform and customize existing tools at the command line level. For this reason, we use CodaLab as the back end for tool execution on the BENTO platform. CodaLab is a cloud-based platform designed for running computational experiments in data-oriented research. In CodaLab, researchers can easily set up a reproducible environment and run arbitrary command line by specifying a docker image and bundle dependencies. In CodaLab, bundles are immutable objects that hold the content of datasets. The output files produced by that command will be saved into a new bundle and can be further passed to down-stream experiments.

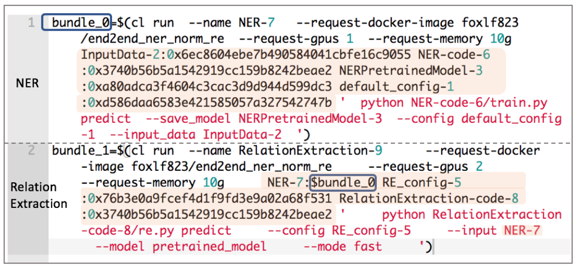
All datasets in BENTO are stored as CodaLab bundles. The tools and pipelines will be compiled into CodaLab commands. Users could submit commands to the CodaLab back end via the web interface. Such design makes the computational results of the BENTO platform reproducible through CodaLab. Since CodaLab will record dependency information in run bundles, it is also easy to recreate the pipeline on our platform from existing result bundles. Using CodaLab as the back end also mitigates the engineering challenges such as job scheduling and data management.



```
[
    config: bundle
  , input: bundle
  , pretrained_model: bundle
] =>

  let
     code = 0x3740b56b5a1542919cc159b8242beae2
  in
  @
  -rquest-docker-image foxlf823/end2end_ner_norm_re
  -request-gpus 1
  -request-memory 10g
  #
  python $code/train.py predict
  -save_model $pretrained_model
  -config $config
  -input_data $input
  @
```

Figure 3: The CodaLang expression for the tool NER in Figure 2. The expression can be roughly split into three sections indicated by the dashed squares. The first section declares the arguments of this tool. As seen, the tool takes three bundles as inputs: *config*, *input* and *pretrained_model*. The second section declares a constant *code* which is initialized with an existing bundle. The third section is a string template for generating the CodaLab command.



```
bundle_0=$(cl run  --name NER-7   --request-docker-image foxlf823
    /end2end_ner_norm_re  --request-gpus 1  --request-memory 10g
    InputData-2:0x6ec8604ebe7b490584041cbfe16c9055 NER-code-6
    :0x3740b56b5a1542919cc159b8242beae2 NERPretrainedModel-3
    :0xa80adca3f4604c3cac3d9d944d599dc3 default_config-1
    :0xd586daa6583e421585057a327542747b '  python NER-code-6/train.py
    predict  --save_model NERPretrainedModel-3  --config default_config
    -1  --input_data InputData-2  ')
bundle_1=$(cl run  --name RelationExtraction-9      --request-docker
    -image foxlf823/end2end_ner_norm_re     --request-gpus 2
    --request-memory 10g      NER-7:$bundle_0 RE_config-5
    :0x76b3e0a9fcef4d1f9fd3e9a02a68f531 RelationExtraction-code-8
    :0x3740b56b5a1542919cc159b8242beae2 '    python RelationExtraction
    -code-8/re.py predict     --config RE_config-5     --input NER-7
      --model pretrained_model     --mode fast     ')
```

Figure 4: The CodaLab commands generated from the pipeline in Figure 2. Two CodaLab commands are generated based on two steps in the pipeline, namely NER and relation extraction. The bundle dependency information is highlighted in orange and the shell commands are colorized in red. The results in the first step are saved in the variable *bundle_0* (circled in blue squares), which is used as a bundle dependency in the command of the second step.

3.3 BENTO Back End

The BENTO back end is for storing tools and generating CodaLab commands from the pipeline graphs.

3.3.1 CodaLang: A Tool Configuration Language

The tools in BENTO are described via our custom language called *CodaLang*[4] It acts as an intermediate layer between the web interface and CodaLab. It has a succinct syntax for specifying the interfaces of a tool, i.e. the inputs and outputs. It also provides a string template mechanism for creating CodaLab commands from input arguments. For example, the CodaLang expression for the node NER in Figure 2 is shown in Figure 3.

The configuration is composed of three sections which are highlighted with dotted squares. The first section declares the arguments of the tool, corresponding to the three input ports of the node. The second section creates a constant variable *code* which is assigned an existing bundle. The third section is a string template for generating the CodaLab command. It includes execution options (e.g., *request-docker-image*) and tool bash commands. The template variables are circled by the squares in the same color with their declarations. Once the values of the tool arguments are determined, a CodaLab command can be easily generated based on the command template. The run bundle created by the command will be used as results and can be passed on to down-stream tools in the pipeline. Through CodaLang, users can easily modify existing tools or create their own tools. The tool configuration can also be automatically generated from the dependency information of a bundle.

[4]A thorough introduction can be found at https://github.com/jyh1/codalang .

3.3.2 Pipeline Execution

We have described how BENTO transforms a single tool to a CodaLab command. In this section, we will describe how BENTO transforms a tool pipeline into multiple CodaLab commands. In a tool pipeline, tools are connected together to form a directed acyclic graph. During execution, tools are transformed to CodaLab commands according to their topological order in the graph. Take the pipeline in Figure 2 as an example, its corresponding CodaLab commands shown in Figure 4.

As shown in Figure 4, the bundle dependency information is highlighted in orange and the shell commands are colorized in red. The two CodaLab commands correspond to the two tool nodes in the pipeline of Figure 2. The first command is generated from the tool NER based on its tool configuration in Figure 3. The results of this command are saved in the variable *bundle_0*, which will be employed as a bundle dependency in the command of the tool for relation extraction. The web interface takes the responsibility of submitting the generated commands to CodaLab. When the pipeline begins to run, the worksheet panel will display the information of the newly created run bundles.

4 Tools Integrated in BENTO

In this section, we list the tools that have already been integrated to our platform, including:

- Hypoglycemic Event Detection (Jin et al., 2019): Hypoglycemic events are common and potentially dangerous conditions among patients being treated for diabetes. This tool can be used to automatically detect hypoglycemic events from EHR notes.

- Clinical Entity Recognition (Li et al., 2018): This tool has been built to recognize 9 types of clinical entities such as medications, indications and adverse drug events (ADEs).

- Clinical Relation Extraction (Li et al., 2018): This tool is able to extract 7 types of relations between clinical entities such as medications and their durations, dosages and frequencies.

- Disease Name Normalization (Li et al., 2019): This tool can be used to normalize disease names to some controlled vocabularies such SNOMED[5] and MEDIC (Davis et al., 2012).

- De-identification: This tool is able to recognize 18 types of protected health information that needs to be removed to de-identify patient notes. We employed BERT (Devlin et al., 2019) to build a de-identification model whose performance is comparable with the state-of-the-art system (Dernoncourt et al., 2017).

We provide examples and instructions to use these tools on the demo page of our platform. For convenience, these tools all take plain text files as inputs and have the pre-processing and tokenization components built-in. In the future, we will integrate stand-alone components dedicated for pre-processing and tokenization to BENTO which can be shared by different application tools. We also plan to incorporate more NLP tools developed by our group(Rumeng et al., 2017; Rawat et al., 2019; Lalor et al., 2019; Zheng and Yu, 2018).

[5]https://www.snomed.org

5 Conclusion

In this paper, we have described the design of the workflow management platform BENTO. To the best of our knowledge, BENTO represents the first web-based workflow management platform for NLP research. Using BENTO, researchers can make use of existing tools or define their own tools. Computational pipelines can be configured through a web-based user-interface and then automatically executed on CodaLab. BENTO includes a number of clinical NLP tools to facilitate the process of EHR notes. A demo of our platform is available at bio-nlp.org/bentodemo/.

References

Enis Afgan, Dannon Baker, Bérénice Batut, Marius Van Den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, et al. 2018. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic acids research*, 46(W1):W537–W544.

TM Apache cTAKES. 2018. clinical text analysis knowledge extraction system.

Hyejin Cho, Wonjun Choi, and Hyunju Lee. 2017. A method for named entity normalization in biomedical articles: application to diseases and plants. *BMC bioinformatics*, 18(1):451.

Allan Peter Davis, Thomas C Wiegers, Michael C Rosenstein, and Carolyn J Mattingly. 2012. Medic: a practical disease vocabulary used at the comparative toxicogenomics database. *Database*, 2012.

Franck Dernoncourt, Ji Young Lee, Ozlem Uzuner, and Peter Szolovits. 2017. De-identification of patient notes with recurrent neural networks. *Journal of the American Medical Informatics Association*, 24(3):596–606.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Mennatallah El-Assady, Wolfgang Jentner, Fabian Sperrle, Rita Sevastjanova, Annette Hautli, Miriam Butt, and Daniel Keim. 2019. lingvis. io-a linguistic visual analytics framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18.

Nitin Hardeniya, Jacob Perkins, Deepti Chopra, Nisheeth Joshi, and Iti Mathur. 2016. *Natural Language Processing: Python and NLTK*. Packt Publishing Ltd.

Bin He, Yi Guan, and Rui Dai. 2019. Classifying medical relations in clinical text via convolutional neural networks. *Artificial intelligence in medicine*, 93:43–49.

Abhyuday N Jagannatha and Hong Yu. 2016. Structured prediction models for rnn based sequence labeling in clinical text. In *Proceedings of the conference on empirical methods in natural language processing. conference on empirical methods in natural language processing*, volume 2016, page 856. NIH Public Access.

Yonghao Jin, Fei Li, Varsha G Vimalananda, and Hong Yu. 2019. Automatic Detection of Hypoglycemic Events From the Electronic Health Record Notes of Diabetes Patients: Empirical Study. *JMIR medical informatics*, 7(4):e14340.

John P Lalor, Beverly Woolf, and Hong Yu. 2019. Improving electronic health record note comprehension with noteaid: Randomized trial of electronic health record note comprehension interventions with crowdsourced workers. *Journal of medical Internet research*, 21(1):e10793.

Fei Li, Yonghao Jin, Weisong Liu, Bhanu Pratap Singh Rawat, Pengshan Cai, and Hong Yu. 2019. Fine-Tuning Bidirectional Encoder Representations From Transformers (BERT)-Based Models on Large-Scale Electronic Health Record Notes: An Empirical Study. *JMIR medical informatics*.

Fei Li, Weisong Liu, and Hong Yu. 2018. Extraction of Information Related to Adverse Drug Events from Electronic Health Record Notes: Design of an End-to-End Model Based on Deep Learning. *JMIR medical informatics*.

Haodi Li, Qingcai Chen, Buzhou Tang, Xiaolong Wang, Hua Xu, Baohua Wang, and Dong Huang. 2017. Cnn-based ranking for biomedical entity normalization. *BMC bioinformatics*, 18(11):79–86.

Zengjian Liu, Buzhou Tang, Xiaolong Wang, and Qingcai Chen. 2017. De-identification of clinical notes via recurrent neural network and conditional random field. *Journal of biomedical informatics*, 75:S34–S42.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Thomas Morton, Joern Kottmann, Jason Baldridge, and Gann Bierner. 2005. Opennlp: A java-based nlp toolkit. In *Proc. EACL*.

Xipeng Qiu, Qi Zhang, and Xuan-Jing Huang. 2013. Fudannlp: A toolkit for chinese natural language processing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 49–54.

Rafal Rak, Andrew Rowley, William Black, and Sophia Ananiadou. 2012. Argo: an integrative, interactive, text mining-based workbench supporting curation. *Database*, 2012.

Bhanu Pratap Singh Rawat, Fei Li, and Hong Yu. 2019. Naranjo question answering using end-to-end multi-task learning model. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2547–2555.

Li Rumeng, N Jagannatha Abhyuday, and Yu Hong. 2017. A hybrid neural network model for joint prediction of presence and period assertions of medical events in clinical notes. In *AMIA Annual Symposium Proceedings*, volume 2017, page 1149. American Medical Informatics Association.

Ergin Soysal, Jingqi Wang, Min Jiang, Yonghui Wu, Serguei Pakhomov, Hongfang Liu, and Hua Xu. 2018. Clamp–a toolkit for efficiently building customized clinical natural language processing pipelines. *Journal of the American Medical Informatics Association*, 25(3):331–336.

Kai Xu, Zhanfan Zhou, Tianyong Hao, and Wenyin Liu. 2017. A bidirectional lstm and conditional random fields approach to medical named entity recognition. In *International Conference on Advanced Intelligent Systems and Informatics*, pages 355–365. Springer.

S Yu and T Cai. 2013. Nile: fast natural language processing for electronic health records. *Preprint at https://arxiv. org/abs/1311.6063*.

Jiaping Zheng and Hong Yu. 2018. Assessing the readability of medical documents: a ranking approach. *JMIR medical informatics*, 6(1):e17.