A Data-Centric Framework for Composable NLP Workflows

Zhengzhong Liu[1,2,*], Guanxiong Ding[2], Avinash Bukkittu[2], Mansi Gupta[2], Pengzhi Gao[2], Atif Ahmed[2], Shikun Zhang[1], Xin Gao[1,2], Swapnil Singhavi[2], Linwei Li[1], Wei Wei[1], Zecong Hu[1], Haoran Shi[1], Xiaodan Liang[3], Teruko Mitamura[1], Eric P. Xing[1,2], and Zhiting Hu[1,4]

[1]Carnegie Mellon University [2]Petuum Inc. [3]Sun Yat-sen University [4]UC San Diego

[*]hectorzliu@gmail.com

Abstract

Empirical natural language processing (NLP) systems in application domains (e.g., healthcare, finance, education) involve interoperation among multiple components, ranging from data ingestion, human annotation, to text retrieval, analysis, generation, and visualization. We establish a unified open-source framework to support fast development of such sophisticated NLP workflows in a composable manner. The framework introduces a uniform data representation to encode heterogeneous results by a wide range of NLP tasks. It offers a large repository of processors for NLP tasks, visualization, and annotation, which can be easily assembled with full interoperability under the unified representation. The highly extensible framework allows plugging in custom processors from external off-the-shelf NLP and deep learning libraries. The whole framework is delivered through two modularized yet integratable open-source projects, namely *Forte*[1] (for workflow infrastructure and NLP function processors) and *Stave*[2] (for user interaction, visualization, and annotation).

1 Introduction

Natural language processing (NLP) techniques are playing an increasingly central role in industrial applications. A real-world NLP system involves a wide range of NLP tasks that interoperate with each other and interact with users to accomplish complex workflows. For example, in an assistive medical system for diagnosis (Figure 4), diverse text *analysis* tasks (e.g., named entity recognition, relation extraction, entity coreference) are performed to extract key information (e.g., symptoms, treatment history) from clinical notes and link to knowledge bases; a medical practitioner could select any extracted entity to *retrieve* similar past cases for reference; text *generation* techniques are used to produce summaries from diverse sources.

To develop domain-specific NLP systems fast, it is highly desirable to have a unified open-source framework that supports: (1) seamless integration and interoperation across NLP functions ranging from text analysis to retrieval to generation; (2) rich user interaction for data visualization and annotation; (3) extensible plug-ins for customized components; and (4) highly reusable components.

A wealth of NLP toolkits exist (§4), such as spaCy (Honnibal and Montani, 2017), DKPro (Eckart de Castilho and Gurevych, 2014), CoreNLP (Manning et al., 2014), for pipelining multiple NLP functions; BRAT (Stenetorp et al., 2012) and YEDDA (Yang et al., 2018) for annotating certain types of data. None of them have addressed all the desiderata uniformly. Combining them for a complete workflow requires non-trivial effort and expertise (e.g., ad-hoc gluing code), posing challenges for maintenance and upgrading.

We introduce a new unified framework to support complex NLP workflows that involve text data ingestion, analysis, retrieval, generation, visualization, and annotation. The framework provides an infrastructure to simply plug in arbitrary NLP functions and offers pre-built and reusable components to build desired workflows. Importantly, the framework is designed to be extensible, allowing users to write custom components (e.g., specialized annotation interfaces) or wrap other existing libraries (e.g., Hu et al., 2019; Wolf et al., 2019) easily.

The framework's design is founded on a data-centric perspective. We design a universal text data representation that can encode diverse input/output formats of various NLP tasks uniformly. Each component ("*processor*") in the workflow fetches relevant parts of data as inputs, and passes its results to subsequent processors by adding the results to

[1]https://github.com/asyml/forte
[2]https://github.com/asyml/stave

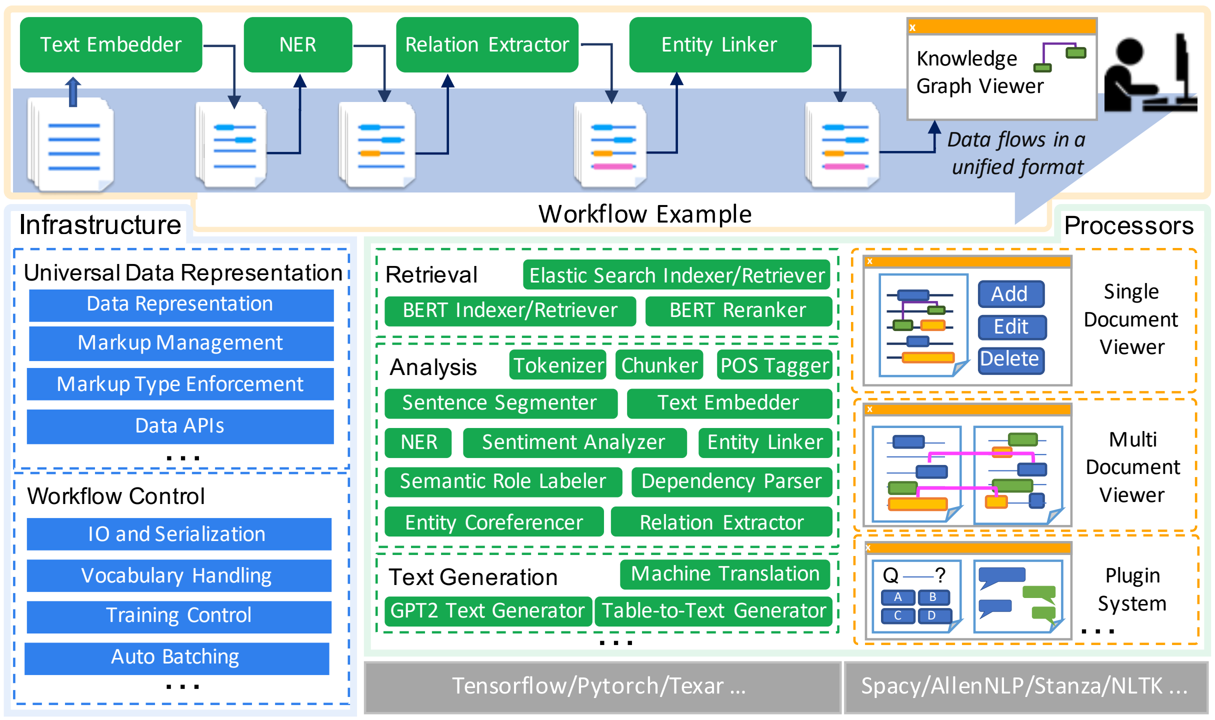


Figure 1: Stack of the data-centric framework for NLP workflows, including workflow *infrastructure*, and *processors* for NLP tasks and interactions (e.g., visualization, annotation). Different processors are composed together with the infrastructure APIs to form an arbitrary complex workflow. The example workflow transforms an unstructured text corpus into a knowledge graph through a series of NLP functions.

the data flow (Figure 1). In this way, different processors are properly decoupled, and each is implemented with a uniform interface without the need of accommodating other processors. Visualization and annotation are also abstracted as standalone components based on the data representation.

We demonstrate two case studies on using the framework to build a sophisticated assistive medical workflow and a neural-symbolic hybrid chatbot.

2 Data-Centric NLP Framework

Figure 1 shows the stack of the framework, consisting of several major parts: **(1)** We first introduce the underlying infrastructure (§2.1), in particular, a universal representation scheme for heterogeneous NLP data. The highly-organized unified representation plays a key role in supporting composable NLP workflows, which differentiates our framework from prominent toolkits such as CoreNLP (Manning et al., 2014), spaCy (Honnibal and Montani, 2017), and AllenNLP (Gardner et al., 2018). We then introduce a range of functionalities that enable the convenient use of the symbolic data/features in neural modeling, which are not available in traditional NLP workflow toolkits such as DKPro (Eckart de Castilho and Gurevych, 2014). **(2)** §2.2 describes how processors for various NLP tasks can be developed with a uniform interface, and can be simply plugged into a complex workflow. **(3)** finally, the human interaction part offers rich composable processors for visualization, annotation, and other forms of interactions.

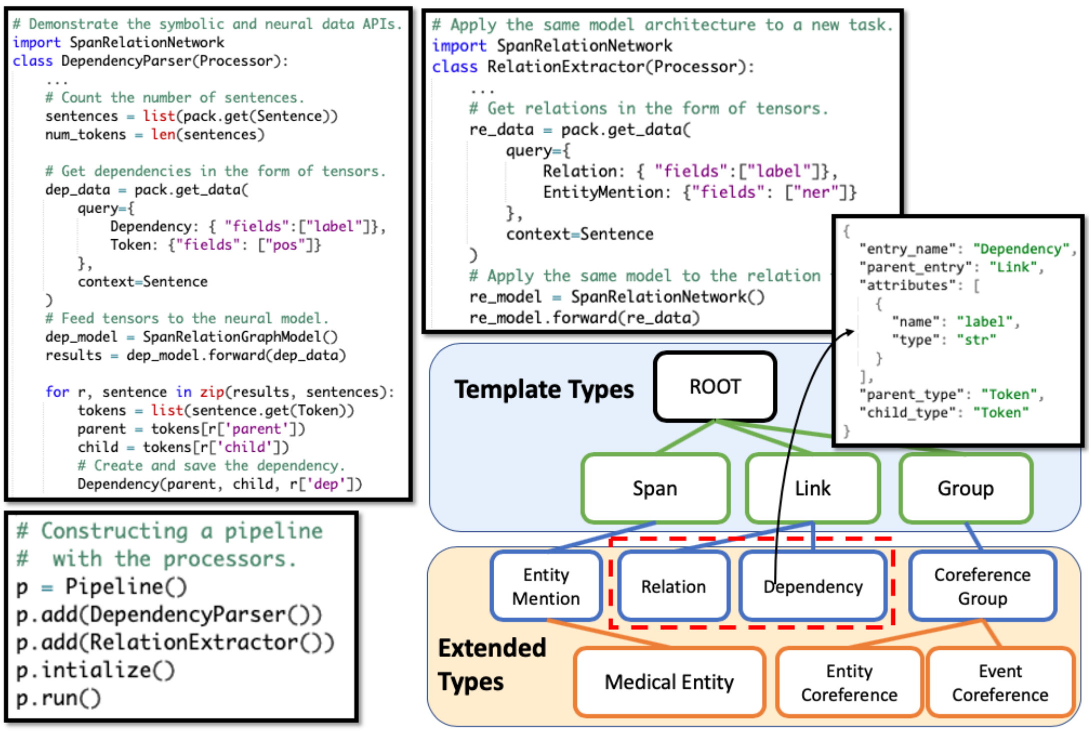
2.1 Infrastructure

2.1.1 Universal Data Representation

NLP data primarily consists of two parts: the raw *text source* and the structured *markups* on top of it (see Figure 3 for an example). The markups represent the information overlaid on the text, such as part-of-speech tags, named entity mentions, dependency links, and so forth. NLP tasks are to produce desired text or markups as output, based on vastly different input information and structures,

To enable full interoperation among distinct tasks, we summarize the underlying commonalities between the myriad formats across different NLP tasks, and develop a universal data representation encapsulating information uniformly. The representation scheme defines a small number of *template* data types with high-level abstraction, which can be further extended to encode domain-specific data.

Template data types: We generalize the previous UIMA representation scheme (Götz and Suhre, 2004) to cover the majority of common NLP



```
# Demonstrate the symbolic and neural data APIs.
import SpanRelationNetwork
class DependencyParser(Processor):
    ...
    # Count the number of sentences.
    sentences = list(pack.get(Sentence))
    num_tokens = len(sentences)

    # Get dependencies in the form of tensors.
    dep_data = pack.get_data(
        query={
            Dependency: { "fields":["label"]},
            Token: {"fields": ["pos"]}
        },
        context=Sentence
    )
    # Feed tensors to the neural model.
    dep_model = SpanRelationGraphModel()
    results = dep_model.forward(dep_data)

    for r, sentence in zip(results, sentences):
        tokens = list(sentence.get(Token))
        parent = tokens[r['parent']]
        child = tokens[r['child']]
        # Create and save the dependency.
        Dependency(parent, child, r['dep'])
```

```
# Apply the same model architecture to a new task.
import SpanRelationNetwork
class RelationExtractor(Processor):
    ...
    # Get relations in the form of tensors.
    re_data = pack.get_data(
        query={
            Relation: { "fields":["label"]},
            EntityMention: {"fields": ["ner"]}
        },
        context=Sentence
    )
    # Apply the same model to the relation
    re_model = SpanRelationNetwork()
    re_model.forward(re_data)
```

```
{
  "entry_name": "Dependency",
  "parent_entry": "Link",
  "attributes": [
    {
      "name": "label",
      "type": "str"
    }
  ],
  "parent_type": "Token",
  "child_type": "Token"
}
```

```
# Constructing a pipeline
#  with the processors.
p = Pipeline()
p.add(DependencyParser())
p.add(RelationExtractor())
p.intialize()
p.run()
```

Figure 2: **Top Left**: A dependency parser processor that calls a neural model and save the results; **Top Right**: A relation extractor can use the same model architecture. **Bottom Left**: A pipeline can be constructed by simply adding processors. **Bottom Right**: Example data types offered by the framework or customized by users. `Relation` and `Dependency` both extends `Link`. Definition of `dependency` is done through a simple JSON configuration.

markups. This results in three template data types, each of which contains a couple of attributes.

- `Span` contains two integer attributes, *begin* and *end*, to denote the offsets of a piece of text. This type can mark tokens, entity mentions, and etc.
- `Link` defines a pair of (*parent*, *child*) which are pointers to other markups, to mark dependency arcs, semantic roles, entity relations, etc.
- `Group` has a *members* attribute, which is a collection of markups. This type can mark coreference clusters, topical clusters, etc.

Extended data types: In order to encode more specific information, each of the template data types can be extended by adding new attributes. For example, the framework offers over 20 extended types for commonly used NLP concepts, such as `Token` and `EntityMention`. Moreover, users can easily add *custom* data types through simple JSON definitions (Figure 2) to fulfill specific needs, such as `MedicalEntity` that extends `EntityMention` with more attributes like patient IDs. Once a new data type is defined, rich data operations (e.g., structured access) as below are automatically enabled for the new type.

Flexible Data Sources: Modern NLP systems face challenges imposed by the volume, veracity and velocity of data. To cope with these, the system is designed with customizable and flexible data sources that embrace technologies such as Indexing (e.g. Elastic Search (Elastic.co)), Databases (e.g. Sqlite), Vector Storage (e.g. Faiss (Johnson et al., 2017)). Users are free to implement flexible "Reader" interface to ingest any source of data.

2.1.2 Facilitation for Neural Modeling

The framework provides extensive functionalities for effortless integration of the above symbolic data representation with tensor-based neural modeling.

Neural representations. All data types are associated with an optional *embedding* attribute to store continuous neural representations. Hence, users can easily access and manipulate the embeddings of arbitrary markups (e.g., entity, relation) extracted from neural models like word2vec (Mikolov et al., 2013) and BERT (Devlin et al., 2019). The system also supports fast embedding indexing and lookup with embedding storage systems such as Faiss (Johnson et al., 2017).

Rich data operations: auto-batching, structured access, etc. Unified data representation enables a rich array of operations to support different data usage, allowing users to access any information in a structured manner. Figure 2 (top left) shows API calls that get all dependency links in a sentence. Utilities such as *auto-batching* and *auto-padding* help aggregates relevant information (e.g., event embeddings) from individual data instances into tensors, which are particularly useful for neural modeling on GPUs.

Neural-symbolic hybrid modeling. Unified data representations and rich data operations make it convenient to support hybrid modeling using both neural and symbolic features. Take retrieval for example, the framework offers retrieval processors (§2.2) that retrieve a coarse-grained candidate set with symbolic features (e.g., TF.IDF) first, and then refine the results with more expensive embedding-based re-ranking (Nogueira and Cho, 2019). Likewise, fast embedding based search is facilitated with the Faiss library (Johnson et al., 2017).

Shared modeling approaches. The uniform input/output representation for NLP tasks makes it easy to share the modeling approaches across diverse tasks. For example, similar to Jiang et al. (2020), all tasks involving the `Span` and `Link` data types as outputs (e.g., dependency parsing, relation extraction, coreference resolution) can potentially use the exact same neural network architecture for modeling. Further with the standardized APIs of our framework, users can spawn models for all such tasks using the same code with minimal edits. Top right of Figure 2 shows an example where the same relation extractor is implemented with dependency parser for a new task, and the only difference lies in accessing different data features.

2.2 Processors

Universal data representation enables a uniform interface to build processors for different NLP tasks. Most notably, *interoperation* across processors supported by the system abstraction allows each processor to interacts with each other via the data flow.

Each processor takes uniformly represented data as inputs and performs arbitrary actions on them. A processor can edit text source (e.g., language generation), add additional markups (e.g., entity detection), or produce side effects (e.g., writing data to disk). Top left of Figure 2 shows the common structure of a processor, that fetches relevant information from the data pack with high-level APIs, performs operations such as neural model inference, writes results back to the data pack and pass them over to subsequent processors. Top right shows the simple API used for plugging processors into the workflow.

A comprehensive repository of pre-built processors. With the standardized concept-level APIs for NLP data management, users can easily develop any desired processors. One can wrap existing models from external libraries by conforming to the simple interfaces. Moreover, we offer a large set of pre-built processors for various NLP tasks, ranging from text retrieval, to analysis and generation. Figure 1 lists a subset of processors.

2.3 Visualization, Annotation, & Interaction

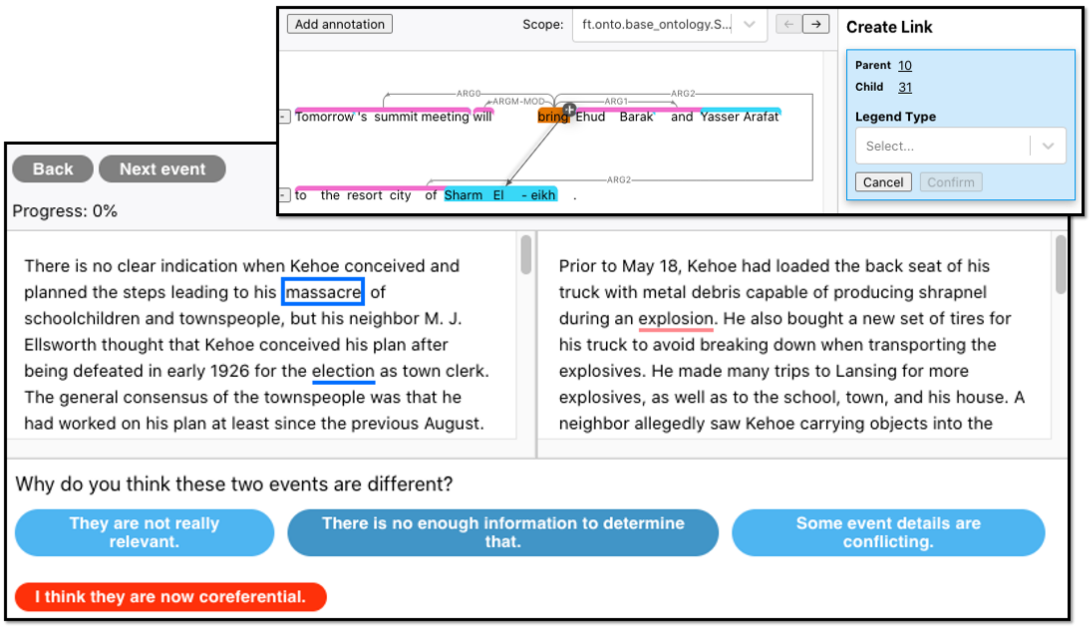
The interfaces for visualization and annotation are implemented as standalone components and designed for different data types.

Single document viewer. We provide a single document interface (Figure 3) that renders the template types. For example, Spans are shown by colored highlights, Links are shown as connectors between the spans. A user can create new spans, add links, create groups, or edit the attributes through intuitive interfaces.

Multi document viewer. *Stave* currently supports a two-document interface. Users can create links across documents. The bottom of Figure 3 shows an example of annotating event coreference. The system is suggesting an event coreference pair and asking for the annotator's decision.

Customization with plugins. While default interfaces support a wide range of tasks, users can create customized interfaces to meet more specific needs. We build a system that can quickly incorporate independently-developed plugins, such as a plugin for answering multiple-choice questions, or a dialogue interface for chat-bots. Some pre-built plugins are showcased in Figure 4. Additionally, the layout can be customized to display specific UI elements, allowing for greater flexibility to use plugins together.

Human-machine collaboration. Universal data representation across all modules not only enhances interoperation between components, but also allows machines and humans to collaborate in a workflow. Human-interactive components can be integrated at any part of the workflow for visualization/reviewing to produce a downstream system that combines the advantages of humans and machines. Machine-assisted annotation can be undertaken straightforwardly: the annotation system simply ingests the data produced by a back-end processor (Figure 3).


Add annotation
Scope: ft.onto.base_ontology.S...
Create Link
Parent 10
Child 31
Legend Type
Select...
Cancel
Confirm
ARG0
ARGM-MOD
ARG1
ARG2
ARG2
Back
Next event
Progress: 0%
They are not really relevant.
There is no enough information to determine that.
Some event details are conflicting.
I think they are now coreferential.

Tomorrow 's summit meeting will bring Ehud Barak and Yasser Arafat to the resort city of Sharm El - eikh .

There is no clear indication when Kehoe conceived and planned the steps leading to his massacre of schoolchildren and townspeople, but his neighbor M. J. Ellsworth thought that Kehoe conceived his plan after being defeated in early 1926 for the election as town clerk. The general consensus of the townspeople was that he had worked on his plan at least since the previous August.

Prior to May 18, Kehoe had loaded the back seat of his truck with metal debris capable of producing shrapnel during an explosion. He also bought a new set of tires for his truck to avoid breaking down when transporting the explosives. He made many trips to Lansing for more explosives, as well as to the school, town, and his house. A neighbor allegedly saw Kehoe carrying objects into the

Why do you think these two events are different?

Figure 3: **Top**: Screenshot of the single doc interface shows predicates, entity mentions and semantic role links of one sentence. A new link is being created from “bring” to “Sharm Ei-eikh”. **Bottom**: Screenshot of the two document interface for annotating event coreference. The system is suggesting a potential coreference pair. The interfaces are rendered based on the data types. Users can customize the interface to use different UI components.

3 Case Studies

3.1 A Clinical Information Workflow

We demonstrate an information system for clinical diagnosis analysis, retrieval, and user interaction. Figure 4 shows an overview of the system. To build the workflow, we first define domain-specific data types, such as `Clinical Entity Mention`, via JSON config files as shown in Figure 2. We then develop processors for text processing: (1) we create an LSTM-based clinical NER processor (Boag et al., 2015), a Span-Relation model based relation extraction processor (He et al., 2018), and a coreference processor with the End-to-End model (Lee et al., 2017) to extract key information; (2) we build a report generation processor following Li et al. (2019) with extracted mentions and relations; (3) we build a simple keyword based dialogue system for user to interact using natural languages. The whole workflow is implemented with minimal engineering effort. For example, the workflow is assembled with just 20 lines of code; and the IE processors are implemented with around 50 lines of code by reusing libraries and models.

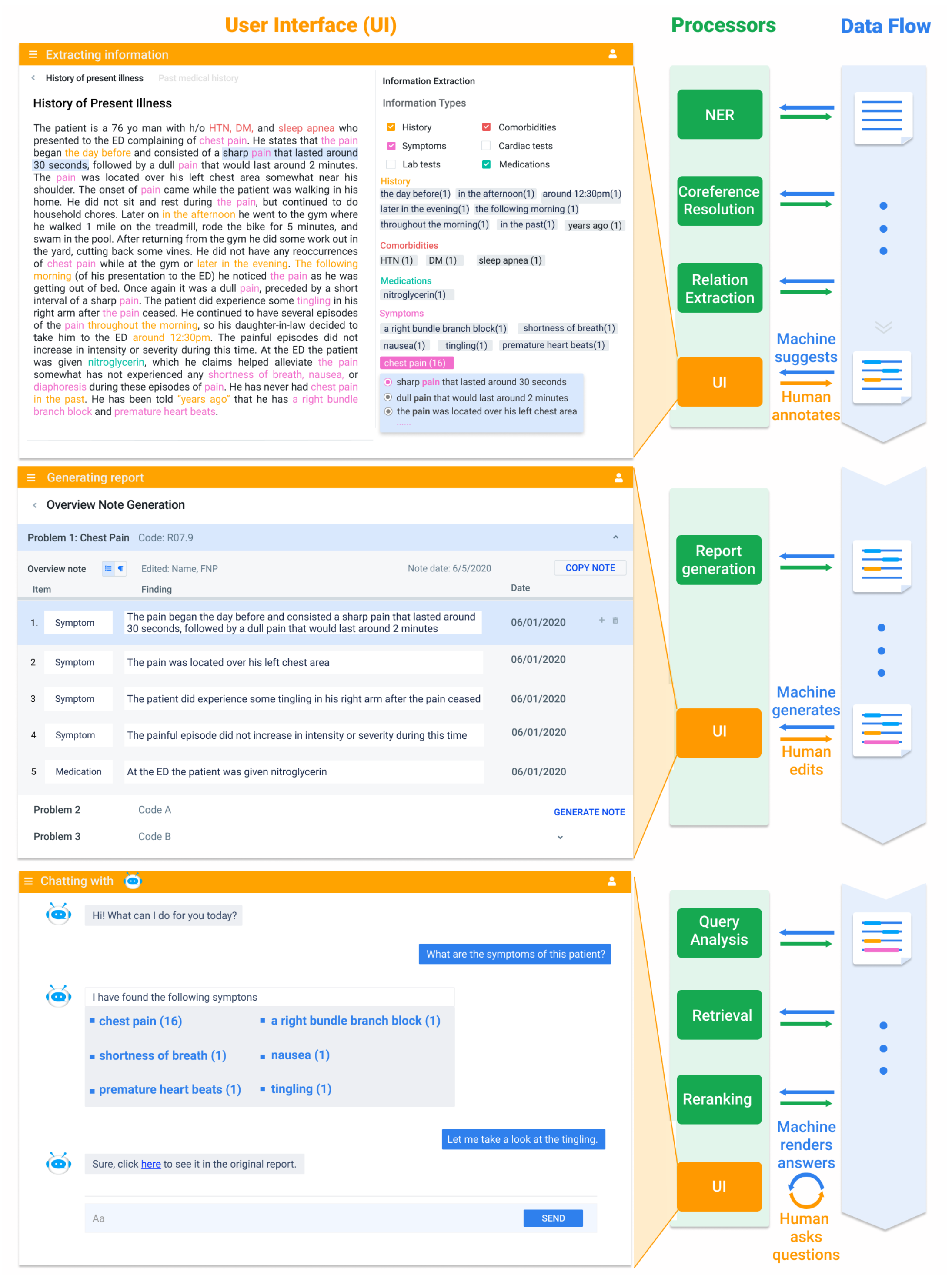
3.2 A ChatBot Workflow

The case study considers the scenario where we have a corpus of movie reviews in English to answer complex queries (e.g., “movies with a positive sentiment starring by a certain actor”) by a German user. The iterative workflow consists of a review retrieval processor based on the hybrid symbolic-neural feature modeling (§2.1.2), an NER processor (Gardner et al., 2018) to find actors and movies from the retrieved reviews, a sentiment processor (Hutto and Gilbert, 2014) for sentence polarity, and an English-German translation processor.

4 Related Work

The framework shares some characteristics with UIMA (Götz and Suhre, 2004) backed systems, such as DKPro (Eckart de Castilho and Gurevych, 2014), ClearTK (Bethard et al., 2014) and cTakes (Khalifa and Meystre, 2015). There are NLP toolboxes like NLTK (Bird and Loper, 2016) and AllenNLP (Gardner et al., 2018), GluonNLP (Guo et al., 2019), NLP pipelines like Stanford CoreNLP (Manning et al., 2014), SpaCy (Honnibal and Montani, 2017), and Illinois Curator (Clarke et al., 2012). As in §2.2, our system develops a convenient scaffold and provides a rich set of utilities to reconcile the benefits of symbolic data system, neural modeling, and human interaction, making it suit for building complex workflows.

Compared to open-source text annotation toolkits, such as Protégé Knowtator (Ogren, 2006), BRAT (Stenetorp et al., 2012), Anafora (Chen and Styler, 2013), GATE (Cunningham et al., 2013), WebAnno (Castilho, 2016), and YEDDA (Yang et al., 2018), our system provides a more flexi-



History of present illness | Past medical history

History of Present Illness

The patient is a 76 yo man with h/o HTN, DM, and sleep apnea who presented to the ED complaining of chest pain. He states that the pain began the day before and consisted of a sharp pain that lasted around 30 seconds, followed by a dull pain that would last around 2 minutes. The pain was located over his left chest area somewhat near his shoulder. The onset of pain came while the patient was walking in his home. He did not sit and rest during the pain, but continued to do household chores. Later on in the afternoon he went to the gym where he walked 1 mile on the treadmill, rode the bike for 5 minutes, and swam in the pool. After returning from the gym he did some work out in the yard, cutting back some vines. He did not have any reoccurrences of chest pain while at the gym or later in the evening. The following morning (of his presentation to the ED) he noticed the pain as he was getting out of bed. Once again it was a dull pain, preceded by a short interval of a sharp pain. The patient did experience some tingling in his right arm after the pain ceased. He continued to have several episodes of the pain throughout the morning, so his daughter-in-law decided to take him to the ED around 12:30pm. The painful episodes did not increase in intensity or severity during this time. At the ED the patient was given nitroglycerin, which he claims helped alleviate the pain somewhat has not experienced any shortness of breath, nausea, or diaphoresis during these episodes of pain. He has never had chest pain in the past. He has been told "years ago" that he has a right bundle branch block and premature heart beats.

Information Extraction

Information Types

✓ History ✓ Comorbidities
✓ Symptoms ☐ Cardiac tests
☐ Lab tests ✓ Medications

History
the day before(1) in the afternoon(1) around 12:30pm(1) later in the evening(1) the following morning (1) throughout the morning(1) in the past(1) years ago (1)

Comorbidities
HTN (1) DM (1) sleep apnea (1)

Medications
nitroglycerin(1)

Symptoms
a right bundle branch block(1) shortness of breath(1) nausea(1) tingling(1) premature heart beats(1) chest pain (16)

- sharp pain that lasted around 30 seconds
- dull pain that would last around 2 minutes
- the pain was located over his left chest area
......

Overview Note Generation

Problem 1: Chest Pain Code: R07.9

Overview note Edited: Name, FNP Note date: 6/5/2020 COPY NOTE

Item		Finding	Date
1.	Symptom	The pain began the day before and consisted a sharp pain that lasted around 30 seconds, followed by a dull pain that would last around 2 minutes	06/01/2020
2	Symptom	The pain was located over his left chest area	06/01/2020
3	Symptom	The patient did experience some tingling in his right arm after the pain ceased	06/01/2020
4	Symptom	The painful episode did not increase in intensity or severity during this time	06/01/2020
5	Medication	At the ED the patient was given nitroglycerin	06/01/2020

Problem 2 Code A GENERATE NOTE

Problem 3 Code B

Hi! What can I do for you today?

What are the symptoms of this patient?

I have found the following symptoms
- chest pain (16)
- a right bundle branch block (1)
- shortness of breath (1)
- nausea (1)
- premature heart beats (1)
- tingling (1)

Let me take a look at the tingling.

Sure, click here to see it in the original report.

Aa SEND

Figure 4: A system for diagnosis analysis and retrieval from clinical notes. The data-centric approach makes it easy to assemble a variety of components and UI elements. Example text was obtained from UNC School of Medicine.

ble experience with customizable plug-ins, extendable data types, and full-fledged NLP support. The Prodigy tool by spaCy is not open-source and supports only pre-defined annotation tasks like NER.

5 Conclusions and Future Work

We present a data-centric framework for building complex NLP workflows with heterogeneous modules. We will continue to improve the framework on other advanced functionalities, such as multi-task learning, joint inference, data augmentation, and provide a broader arsenal of processors to help build better NLP solutions and other data science workflows. We also plan to further facilitate workflow development by providing more flexible and robust data management processors.

References

Steven Bethard, Philip Ogren, and Lee Becker. 2014. Cleartk 2.0: Design patterns for machine learning in uima. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 3289–3293, Reykjavik, Iceland. European Language Resources Association (ELRA).

Steven Bird and Edward Loper. 2016. NLTK : The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, March, pages 63–70.

William Boag, Kevin Wacome, Tristan Naumann, and Anna Rumshisky. 2015. CliNER: A Lightweight Tool for Clinical Named Entity Recognition. In *AMIA Joint Summits on Clinical Research Informatics*.

Richard Eckart De Castilho. 2016. A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 76–84.

Wei-te Chen and Will Styler. 2013. Anafora : A Web-based General Purpose Annotation Tool. In *Proceedings of the NAACL HLT 2013 Demonstration Session*, June, pages 14–19.

James Clarke, Vivek Srikumar, Mark Sammons, and Dan Roth. 2012. An NLP curator (or: How I learned to stop worrying and love NLP pipelines). *Proceedings of the 8th International Conference on Language Resources and Evaluation, LREC 2012*, pages 3276–3283.

Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. 2013. Getting More Out of Biomedical Documents with GATE ' s Full Lifecycle Open Source Text Analytics. *PLOS Computational Biology*, 9(2).

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.

Richard Eckart de Castilho and Iryna Gurevych. 2014. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–.

Elastic.co. Elasticsearch.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6.

T. Götz and O. Suhre. 2004. Design and implementation of the UIMA common analysis system. *IBM Systems Journal*, 43(3):476–789.

Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, and Shuai Zheng. 2019. GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing. Technical report.

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly Predicting Predicates and Arguments in Neural Semantic Role Labeling. In *ACL 2018*.

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

Zhiting Hu, Haoran Shi, Bowen Tan, Wentao Wang, Zichao Yang, Tiancheng Zhao, Junxian He, Lianhui Qin, Di Wang, et al. 2019. Texar: A modularized, versatile, and extensible toolkit for text generation. In *ACL 2019, System Demonstrations*.

Clayton J. Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *ICWSM*.

Zhengbao Jiang, Wei Xu, Jun Araki, and Graham Neubig. 2020. Generalizing Natural Language Analysis through Span-relation Representations. Technical report.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Abdulrahman Khalifa and Stephane Meystre. 2015. Adapting existing natural language processing resources for cardiovascular risk factors identification in clinical notes. *Journal of Biomedical Informatics*, 58S.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In *EMNLP 2017*.

Christy Y Li, Xiaodan Liang, Zhiting Hu, and Eric P Xing. 2019. Knowledge-driven Encode, Retrieve, Paraphrase for Medical Image Report Generation. In *AAAI 2019*.

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, Baltimore, Maryland. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Philip V Ogren. 2006. Knowtator : A Protégé plug-in for annotated corpus construction. In *Proceedings of the Human Language Technology Conference of the NAACL*, June, pages 273–275.

Pontus Stenetorp, Sampo Pyysalo, and Goran Topi. 2012. BRAT : a Web-based Tool for NLP-Assisted Text Annotation. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107.

UNC School of Medicine. History and Physical Examination Examples 5.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, pages arXiv–1910.

Jie Yang, Yue Zhang, Linwei Li, and Xingxuan Li. 2018. YEDDA : A Lightweight Collaborative Text Span Annotation Tool. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics-System Demonstrations*, pages 31–36.