CogKTR: A Knowledge-Enhanced Text Representation Toolkit for Natural Language Understanding

Zhuoran Jin\textsuperscript{1,2}, Tianyi Men\textsuperscript{1,2}, Hongbang Yuan\textsuperscript{1,2}, Yuyang Zhou\textsuperscript{2}, Pengfei Cao\textsuperscript{1,2}, Yubo Chen\textsuperscript{1,2}, Zhipeng Xue\textsuperscript{2}, Kang Liu\textsuperscript{1,2,3}, Jun Zhao\textsuperscript{1,2}

\textsuperscript{1} School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
\textsuperscript{2} National Laboratory of Pattern Recognition, Institute of Automation, CAS, Beijing, China
\textsuperscript{3} Beijing Academy of Artificial Intelligence, Beijing, China

\{zhuoran.jin, tianyi.men, hongbang.yuan\}@nlpr.ia.ac.cn, zhouyuyang17@163.com
\{pengfei.cao, yubo.chen, zhipeng.xue, kliu, jzhao\}@nlpr.ia.ac.cn

Abstract

As the first step of modern natural language processing, text representation encodes discrete texts as continuous embeddings. Pre-trained language models (PLMs) have demonstrated strong ability in text representation and significantly promoted the development of natural language understanding (NLU). However, existing PLMs represent a text solely by its context, which is not enough to support knowledge-intensive NLU tasks. Knowledge is power, and fusing external knowledge explicitly into PLMs can provide knowledgeable text representations. Since previous knowledge-enhanced methods differ in many aspects, making it difficult for us to reproduce previous methods, implement new methods, and transfer between different methods. It is highly desirable to have a unified paradigm to encompass all kinds of methods in one framework. In this paper, we propose CogKTR, a knowledge-enhanced text representation toolkit for natural language understanding. According to our proposed Unified Knowledge-Enhanced Paradigm (UniKEP), CogKTR consists of four key stages, including knowledge acquisition, knowledge representation, knowledge injection, and knowledge application. CogKTR currently supports easy-to-use knowledge acquisition interfaces, multi-source knowledge embeddings, diverse knowledge-enhanced models, and various knowledge-intensive NLU tasks. Our unified, knowledgeable and modular toolkit is publicly available at GitHub \textsuperscript{1}, with an online system \textsuperscript{2} and a short instruction video \textsuperscript{3}.

1 Introduction

In modern natural language processing (NLP), texts need to be represented into a machine-readable form. Many work has shown that pre-trained language models (PLMs) (Qiu et al., 2020) can provide powerful distributed representations for natural language texts, leading to great successes on various natural language understanding (NLU) (Wang et al., 2018a) tasks.

Recently, some studies (Manning et al., 2020; Roberts et al., 2020; Penha and Hauff, 2020) have shown that specific knowledge is implicitly stored in the parameters of PLMs. This implicit knowledge is vague so that it is hard to dynamically update this knowledge to satisfy the needs of real-world applications (Yin et al., 2022). Existing PLMs (Peters et al., 2018; Devlin et al., 2019) represent and understand a text solely by its context, which is insufficient to solve knowledge-intensive NLU tasks. These tasks are highly dependent on background knowledge. It is necessary to leverage external knowledge to enhance the text representations explicitly. For word sense disambiguation, synonyms, sense definitions, and other linguistic knowledge play an essential role in identifying the meaning of ambiguous words. For commonsense question answering, commonsense knowledge like structured knowledge graph (KG) triples can enhance the models’ reasoning capacity.

As illustrated above, knowledge-enhanced text representations are essential for NLU tasks, meanwhile, many methods (Wei et al., 2021; Ding et al., 2022; Zhu et al., 2022) have been proposed. However, previous methods differ in many aspects, especially in knowledge acquisition procedure, knowledge representation form, and knowledge fusion approach. These differences make it challenging to reproduce previous methods, implement new methods, and transfer between different methods. So we need a unified paradigm to implement various knowledge-enhanced methods in the same framework. Therefore, designing the framework should consider the following key principles.

First, the process of knowledge acquisition is laborious and complex, including knowledge tag-
ging (e.g., named entity recognition and semantic role labeling), knowledge grounding (e.g., entity linking) and knowledge retrieving (e.g., regular expression matching and SPARQL query). A good framework should let users pay more attention to the details in the models rather than tedious data processing. Second, different knowledge embeddings vary in knowledge sources (e.g., Wikidata (Vrandečić and Krötzsch, 2014) and ConceptNet (Speer et al., 2017)) and knowledge representation algorithms (e.g., TransE (Bordes et al., 2013) and Wikipedia2Vec (Yamada et al., 2020a)). To make rigorous comparisons between them, it is highly desirable to have a toolkit that provides built-in knowledge embeddings. Third, although a lot of knowledge fusion approaches have been proposed, there is still a lack of a comprehensive framework to encompass them. Such a framework should provide knowledgeable text representations which can be directly used in numerous downstream tasks.

To this end, we propose CogKTR, a Knowledge-enhanced Text Representation toolkit for natural language understanding. CogKTR is built on the Unified Knowledge-Enhanced Paradigm (UniKEP), which can be formalized in four stages, including knowledge acquisition, knowledge representation, knowledge injection, and knowledge application. First, knowledge acquisition aims to identify structured information from unstructured texts, then ground them in knowledge sources. Then, knowledge representation can transform knowledge from discrete form to continuous form. Next, knowledge injection, as the most critical stage, combines raw texts and external knowledge for knowledgeable text representation. In the end, knowledge application verifies the effectiveness of knowledge-enhanced methods in downstream tasks.

In detail, CogKTR has the following functions. First, our toolkit provides user-friendly knowledge acquisition interfaces. Users can use our toolkit to enhance the given texts with one click. And we also implement plenty of knowledge-enhanced methods so researchers can quickly reproduce these models. Moreover, CogKTR supports many built-in NLU tasks to evaluate the effectiveness of knowledge-enhanced methods. In our paradigm, users can easily conduct their research via a pipeline. Besides the toolkit, we also release an online CogKTR demo to show the process of knowledge acquisition and the effect of knowledge enhancement.

In summary, the main features and contributions are as follows:

- **Unified.** CogKTR is designed and built on our Unified Knowledge-Enhanced Paradigm, which consists of four stages: knowledge acquisition, knowledge representation, knowledge injection, and knowledge application.

- **Knowledgeable.** CogKTR integrates multiple knowledge sources, including Wikidata, Wikipedia, ConceptNet, WordNet (Miller, 1995) and CogNet (Wang et al., 2021a), and implements a series of knowledge-enhanced methods, such as K-BERT (Liu et al., 2020), SemBERT (Zhang et al., 2020a), QAGNN (Yasunaga et al., 2021), etc.

- **Modular.** CogKTR modularizes our proposed paradigm and consists of Enhancer, Model, Core and Data modules, each of which is highly extensible so that researchers can implement new components easily.

2 Unified Knowledge-Enhanced Paradigm

As mentioned above, it is vital to propose a paradigm that can formalize the knowledge-enhanced process. As shown in Figure 1, our proposed Unified Knowledge-Enhanced Paradigm (UniKEP) consists of four key stages: knowledge acquisition, knowledge representation, knowledge injection and knowledge application. Below are the detailed descriptions of the four stages.

2.1 Knowledge Acquisition

Knowledge acquisition, the first step towards our knowledge-enhanced paradigm, aims at detecting knowledge concealed beneath the raw texts. Details of our implementation of the acquisition process can be found in Section 3.1. The obtained knowledge can be divided into three categories according to the different sources they belong to.

**World Knowledge.** It contains general facts about some particular entities or events. For example, given a sentence “Elmo and Bert read books in the Sesame street library.”, “Elmo”, “Bert” and “Sesame street” can be spotted as entities via named entity recognition. Then, “Bert” can be linked to the target entity “Bert (Sesame Street)” in Wikipedia via entity linking. World knowledge is helpful in many entity-related tasks, such as entity typing, relation extraction and fact verification.
**Linguistic Knowledge.** It refers to the internal syntactic structure and the meaning of words and phrases in the texts. As shown in Figure 1, the dependency tree describes the directed grammatical relations between words and semantic role labeling extracts the predicate-argument structure. Incorporating linguistic knowledge can bring better text representations in downstream tasks like information retrieval and machine reading comprehension.

**Commonsense Knowledge.** It tries to catch implicit facts in our daily life. For example, (Bert, is a type of, fictional character) and (library, is used for, reading) are the commonsense triples extracted from ConceptNet. Current models usually have a poor commonsense awareness, thus leveraging commonsense knowledge can help models gain stronger capability on commonsense reasoning.

### 2.2 Knowledge Representation

The aforementioned knowledge can be represented in two forms, including discrete representation and continuous representation.

**Discrete Representation.** Discrete knowledge is usually represented as texts, triples, subgraphs and symbols. Texts are the most commonly used representation forms, such as descriptions of nodes and relations in KGs or definitions of words in lexicons. Triples describe a particular connection between two nodes in KGs. A subgraph’s topology contributes a lot to the comprehension of the central node. However, discrete knowledge cannot be directly used in deep learning systems and need to be further represented.

**Continuous Representation.** It usually refers to the dense vectors in a unified continuous representation space. The traditional skip-gram model can be used to compute the embeddings of words (Yamada et al., 2020a). Entities and relations in triples can be viewed as translational operations and points from the perspective of conventional knowledge embedding models (Bordes et al., 2013). The continuous representation can be easily fused to models as prior knowledge.

### 2.3 Knowledge Injection

Injecting knowledge into original models is vital to the whole paradigm. The injection strategy varies depending on when knowledge is fused into original models. We divide them into three categories: knowledge-enriched input, knowledge-aware architecture and knowledge-assistant training.

**Knowledge-enriched Input.** A typical case of knowledge injection is to combine the input text with the extracted knowledge. Entity descriptions, concepts, brief interpretations and synonyms of the words can all be concatenated together with original texts to form input samples of the model. However, too much knowledge may be noisy. Thus
some attention masks are constructed for the self-attention process in the model. Besides, pretrained knowledge embeddings can be fused to the text representations by direct arithmetic operations.

Knowledge-aware Architecture. In some cases, a certain architecture is designed to encode the extracted knowledge. Graph neural network (GNN) is often used to encode the structured knowledge (Yu et al., 2022). Transformer-like architectures is usually used to deal with textual descriptions (Zhang et al., 2019). Memory network is used to restore learned knowledge embeddings and can be applied to any sequence output (Févry et al., 2020).

Knowledge-assisted Training. Knowledge can also be used to design knowledge-driven training objectives. Entity-level masking masks the entities in a sentence to guide the text representation learning (Sun et al., 2019). Relation prediction requires models to identify the relation between two given entities in order to inject world knowledge (Wang et al., 2021b). Supersense prediction trains the model to classify the masked word’s sense into 45 supersense categories (Levine et al., 2020).

2.4 Knowledge Application

Various downstream NLU tasks can benefit from the knowledge-enhanced models. This subsection presents the definition, application and necessity of the existence of external knowledge of each downstream NLU task.

Text Classification. It is a task to assign labels to language entries like sentences or documents. Sentiment analysis, fact verification, and fake news detection all fall into this category. Fake news detection needs additional knowledge to serve as evidence for better detection (Hu et al., 2021).

Text Matching. It is a task determining whether one sentence is related to another based on semantic meanings and plays a significant role in text entailment and entity disambiguation. For text entailment, knowledge in the two statements can help information flow between them (Jo et al., 2021).

Sequence Labeling. This task is to label each token of the given sentence. Named entity recognition (NER), part-of-speech tagging and semantic role labeling can be viewed as a sequence labeling problem. For example, a preconstructed entity dictionary contributes to recognizing the entity boundary in NER tasks (Zhang and Yang, 2018).

Machine Reading Comprehension. This task is to comprehend a given passage and then answer questions based on it. It can be approximately divided into four different kinds of forms: cloze-style, multi-choice, span extraction and free-form. In open domain QA, knowledge can be beneficial in identifying answers which are not likely lying inside the given context (Yamada et al., 2020b).

3 System Design and Architecture

According to the paradigm mentioned above, we divide CogKTR architecture into four modules. For knowledge acquisition and representation, CogKTR modularizes them as the Enhancer module. To implement knowledge injection and application, we build the Model module to integrate knowledge into models. Considering that the development process is time-consuming, we also design two basic modules, namely Data module and Core module, to accelerate the data processing procedure and improve training efficiency. An overview of CogKTR architecture is shown in Figure 2. In the following, we will introduce these four modules.

3.1 Enhancer

This module is designed for knowledge acquisition and representation to leverage relevant knowledge to enhance raw texts. It can be divided into four steps. Firstly, parse sentences and detect candidate mentions by Tagger. Then, link these mentions to KGs by Linker. Next, search the relevant information in KGs and textual corpus by Searcher. Finally, convert discrete knowledge to dense embeddings in continuous space by Embedder. The specific classes of Enhancer, Tagger, Linker, Searcher and Embedder are represented in Table 1.

Tagger. It is a text annotator to convert unstructured texts into structured knowledge. It can be categorized into two streams according to the existence of external KGs. If corresponding KGs exist, we focus on identifying the locations of knowledge-related text mentions, including words, entities, phrases, etc. They can be linked to KGs, enriching raw sentences with external information. Otherwise, we parse the given sentences to obtain internal syntactic and semantic knowledge, such as part-of-speech tags, dependency trees and semantic role labels. CogKTR contains three external knowledge taggers and three internal knowledge taggers.
**Linker.** It aims to link the candidate mentions detected by the Tagger modules to external KGs. It is an essential bridge between unstructured texts and structured knowledge, where linking methods include entity linking and string matching. Entity linking is based on measuring the similarity between mentions in the texts and entities in KGs and string matching is to find the corresponding nodes in KGs through strict comparison or fuzzy query. We implement three linkers in CogKTR.

**Searcher.** It is to retrieve detailed information about target mentions in KGs (such as Wikipedia, ConceptNet and WordNet), and textual corpus. In this paper, we divide KG-related knowledge into unstructured textual information and structured information. Unstructured textual information includes entity titles, entity descriptions and example sentences, while structured information includes triples, subgraphs and relation paths. As for textual corpus, we use retrieval methods to obtain related texts of the queries. We implement four searchers.

**Embedder.** It is used to embed discrete knowledge into continuous space. We encode KGs as low-dimensional and dense vectors by TransE, Wikipedia2Vec and PLMs, which can be directly injected into deep learning models.

### 3.2 Model

To implement knowledge injection and application, we design the Model module to fuse texts and knowledge acquired from the Enhancer module. For extensibility, we decouple the Model module into T-Model and K-Model. T-Model denotes task-specific models, designed for various downstream tasks. K-Model denotes knowledge-enhanced models, aiming to inject knowledge into PLMs to represent texts. K-Model and T-Model can be combined to realize the application of different knowledge-enhanced models on different downstream tasks.

**T-Model.** This module is used to achieve downstream tasks. It can be classified into seven types: ReadingComprehension, TextClassification, MLM, QuestionAnswering, SequenceLabeling, TextMatching, Disambiguation class.

**K-Model.** This module is responsible for knowledge injection and built on huggingface transformers library (Poerner et al., 2020). It can be divided into two categories: (1) **Input-enhanced models** aim to enrich input texts and constrain attention masks. In terms of input texts, we divide injection into two types, discrete injection and continuous injection. Discrete injection means concatenating raw texts and additional knowledge texts like ESR (Song et al., 2021), K-BERT (Liu et al., 2020), and then feeding into PLMs. Continuous injection refers to converting texts or entities into vectors, such as KT-Emb and KG-Emb (Xu et al., 2021). For attention masks, symbolic knowledge like dependency trees with directed graphs is used to constrain attention masks based on Sg-Net (Zhang et al., 2020b). (2) **Architecture-enhanced models** use additional network architecture to encode knowledge and incorporate knowledge representation into language models. In CogKTR, SAFE (Jiang et al., 2022) is used to encode relation paths by MLP, while RNN is used to capture semantic dependencies.

---

**Figure 2:** The system architecture of CogKTR.
role labeling knowledge like SemBERT (Zhang et al., 2020a). For graph structure knowledge, we implement QAGNN (Yasunaga et al., 2021) and HLG (Li et al., 2022) models with GNN to encode commonsense knowledge and linguistic knowledge.

### 3.3 Data

This module is responsible for data loading and processing procedures. It is composed of Reader and Processor classes. To unify input, we design Reader class to load raw datasets, which inherits from BaseReader class. The Processor class is a data processing component in CogKTR. It is used to build the bridge among models, raw data and enhanced data, which can process raw data and enhanced data into the form required by the models.

### 3.4 Core

It focuses on accelerating the efficiency of model training and evaluation. It contains Trainer, Evaluator, Predictor and Analyzer classes. Trainer class is designed for model training, supporting multi-GPU distributed parallel training and experimental results recording. Evaluator class contains classification metric, regression metric, reading comprehension metric and so on. Predictor class supports various downstream inference tasks with additional knowledge.

### 4 System Usage

In this section, we will give detailed guidelines on how to use CogKTR toolkit and online demo.

#### 4.1 Code Usage

We separate the source code to three main parts: enhancing the given texts with knowledge, constructing a knowledge-aware model and training the model. In Appendix A, Figure 3 shows an example for the usage of our code. We formalize a pipeline for these three steps so users can achieve our Unified Knowledge-Enhanced Paradigm easily. Before processing the input text, users should prepare the corresponding knowledge sources, which will be downloaded automatically. Then, the Reader, Enhancer and Processor class should be initialized to generate the knowledge-enhanced input of the models. Moreover, the T-Model, Metric, Loss and Optimizer class should be initialized before added to Trainer class. Users should initialize the K-Model class as the knowledge-enhanced encoder of the T-Model class.

#### 4.2 Demo Usage

Besides this toolkit, we also release an online demo as shown in Figure 4, 5 and 6. The online demo consists of two parts: knowledge-enhanced text and knowledge-enhanced task. The knowledge-enhanced text part will acquire different types of knowledge in the given sentence, including world, linguistic, and commonsense knowledge. And the knowledge-enhanced task part performs different downstream tasks, including sentiment analysis, text entailment and commonsense reasoning.

### 5 Evaluation

CogKTR aims to support various NLU tasks under a unified paradigm. To demonstrate the effectiveness of knowledge-enhanced methods, we implement several baselines and evaluate them on the corresponding tasks. The evaluation tasks include CommonsenseQA (Talmor et al., 2018) and OpenBookQA (Mihaylov et al., 2018) for commonsense reasoning; LAMA (Petroni et al., 2019) for knowledge probing; SQuAD2.0 (Rajpurkar et al., 2018) for reading comprehension; QNLI and SST-B (Wang et al., 2018b) for text entailment; CoNLL2003 (Sang and De Meulder, 2003) for sequence labeling; SST-2 and SST-5 (Socher et al., 2013) for sentiment analysis; SemCor (Miller et al., 1994) and SemEval (Pradhan et al., 2007) for word sense disambiguation. Reader and Processor classes of these datasets have already been integrated into CogKTR. The experimental results are available at our GitHub 4.

### 6 Conclusion and Future Work

In this paper, we propose CogKTR, a knowledge-enhanced text representation toolkit for natural language understanding. CogKTR is built on our Unified Knowledge-Enhanced Paradigm, which is composed of four stages: knowledge acquisition, knowledge representation, knowledge injection, and knowledge application. In CogKTR, we provide easy-to-use knowledge acquisition interface, off-the-shelf knowledge embeddings, built-in knowledge-enhanced models, and knowledge-intensive NLU tasks. Besides the toolkit, we also release an online demo system. In the future, more knowledge sources, benchmark datasets, and models will be incorporated into CogKTR.

4https://github.com/CogNLP/CogKTR/
Limitations

In this paper, we propose Unified Knowledge-Enhanced Paradigm to formalize the knowledge-enhanced process. However, there are still some limitations in the existing knowledge-enhanced process. We discuss these in detail below.

First, in the knowledge acquisition stage, we should discover knowledge from raw texts via name entity recognition, entity linking, semantic role labeling and other methods. These methods are usually provided by off-the-shelf toolkits, causing inevitable errors. Such noise will affect the performance on downstream tasks. In the future work, we should further study how to eliminate the influence of noise caused by knowledge acquisition.

Second, a vast number of knowledge embedding methods are designed to address knowledge graph completion (KGC), which aims to predict missing links for KGs. These methods only consider the structured information and ignore the valuable textual and logic knowledge in KGs. How to provide more informative knowledge embeddings for knowledge-enhanced models is worth studying.

Finally, we utilize a broad set of downstream tasks to evaluate the knowledge-enhanced models. But better performance does not mean that the model has really learned the knowledge. We should find a better way to probe the knowledge in models and improve the interpretability.

Acknowledgements

We thank all the anonymous reviewers. This work is supported by the National Key Research and Development Program of China (No. 2020AAA0106400), the National Natural Science Foundation of China (No. 61922085, 61976211, 62176257). This work is also supported by the Youth Innovation Promotion Association CAS.

References


Table 1: Specific classes of Enhancer module, contains Tagger, Linker, Searcher and Embedder components.

<table>
<thead>
<tr>
<th>Components</th>
<th>Classes</th>
<th>Functions</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagger</td>
<td>NerTagger</td>
<td>identify entity mention spans</td>
<td>CogIE (Jin et al., 2021)</td>
</tr>
<tr>
<td></td>
<td>ConceptNetTagger</td>
<td>identify concept mention spans</td>
<td>spaCy (Honnibal et al., 2020)</td>
</tr>
<tr>
<td></td>
<td>WordNetTagger</td>
<td>identify candidate texts spans</td>
<td>NLTK (Bird et al., 2009)</td>
</tr>
<tr>
<td></td>
<td>SrlTagger</td>
<td>tag sentences and get semantics labeling</td>
<td>Stanza (Qi et al., 2020)</td>
</tr>
<tr>
<td></td>
<td>SyntaxTagger</td>
<td>parse sentences and get dependency trees</td>
<td>AllenNLP (Gardner et al., 2017)</td>
</tr>
<tr>
<td></td>
<td>WordSegmentationTagger</td>
<td>chinese word segmentation</td>
<td>jieba</td>
</tr>
<tr>
<td>Linker</td>
<td>WikipediaLinker</td>
<td>link entities to Wikipedia</td>
<td>CogIE (Jin et al., 2021)</td>
</tr>
<tr>
<td></td>
<td>ConceptNetLinker</td>
<td>link concepts to ConceptNet</td>
<td>spaCy (Honnibal et al., 2020)</td>
</tr>
<tr>
<td></td>
<td>WordNetLinker</td>
<td>link candidate texts to WordNet</td>
<td>CogIE (Jin et al., 2021)</td>
</tr>
<tr>
<td>Searcher</td>
<td>WikipediaSearcher</td>
<td>query entity titles and text descriptions in Wikipedia</td>
<td>KILT (Bird et al., 2009)</td>
</tr>
<tr>
<td></td>
<td>WikidataSearcher</td>
<td>look up triples and subgraphs in Wikidata</td>
<td>qwikidata</td>
</tr>
<tr>
<td></td>
<td>ConceptNetSearcher</td>
<td>search subgraphs and relation paths in ConceptNet</td>
<td>spaCy (Honnibal et al., 2020)</td>
</tr>
<tr>
<td></td>
<td>WordNetSearcher</td>
<td>synonyms, example sentences, definitions and hypernyms</td>
<td>NLTK (Bird et al., 2009)</td>
</tr>
<tr>
<td>Embedder</td>
<td>WikidataEmbedder</td>
<td>convert Wikidata into continuous knowledge</td>
<td>CogKGE (Jin et al., 2022)</td>
</tr>
<tr>
<td></td>
<td>ConceptNetEmbedder</td>
<td>convert ConceptNet into continuous knowledge</td>
<td>MHGRN (Peng et al., 2020)</td>
</tr>
<tr>
<td></td>
<td>WordNetEmbedder</td>
<td>convert WordNet into continuous knowledge</td>
<td>CogKGE (Jin et al., 2022)</td>
</tr>
</tbody>
</table>

Figure 3: A code example of model training.

```python
import cogktr
import torch
# Load the dataset and construct the vocabulary
reader = cogktr.Reader(data_path)
train_data, dev_data, test_data = reader.read_all()
vocab = reader.read_vocab()

# Enhance the data
enhancer = cogktr.Enhancer(knowledge_graph_path, cache_path, cache_file)
enhanced_train_dict = enhancer.enhance_train(datable=train_data, return_entity_desc=True)
enhanced_dev_dict = enhancer.enhance_dev(datable=dev_data, return_entity_desc=True)
enhanced_test_dict = enhancer.enhance_test(datable=test_data, return_entity_desc=True)

# Process the data with external knowledge
processor = cogktr.Processor(max_token_len=128, vocab=vocab)
train_dataset = processor.process_train(data=train_data, enhanced_dict=enhanced_train_dict)
dev_dataset = processor.process_dev(data=dev_data, enhanced_dict=enhanced_dev_dict)
test_dataset = processor.process_test(data=test_data, enhanced_dict=enhanced_test_dict)

# Construct the knowledge-aware model
k_model = cogktr.KModel(pretrained_model="bert-base-cased")
t_model = cogktr.TModel(k_model, vocab)
metrics = cogktr.Metrics(mode="multi")
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(t_model.parameters())

# Train the knowledge-aware model
trainer = cogktr.Trainer(t_model, train_dataset, dev_dataset, n_epochs=1000,
batch_size=128, loss=loss, optimizer=optimizer, metrics=metrics)
trainer.train()
```
Figure 4: A demo example of world knowledge acquisition.

Figure 5: A demo example of linguistic knowledge acquisition.

Figure 6: A demo example of commonsense reasoning task.