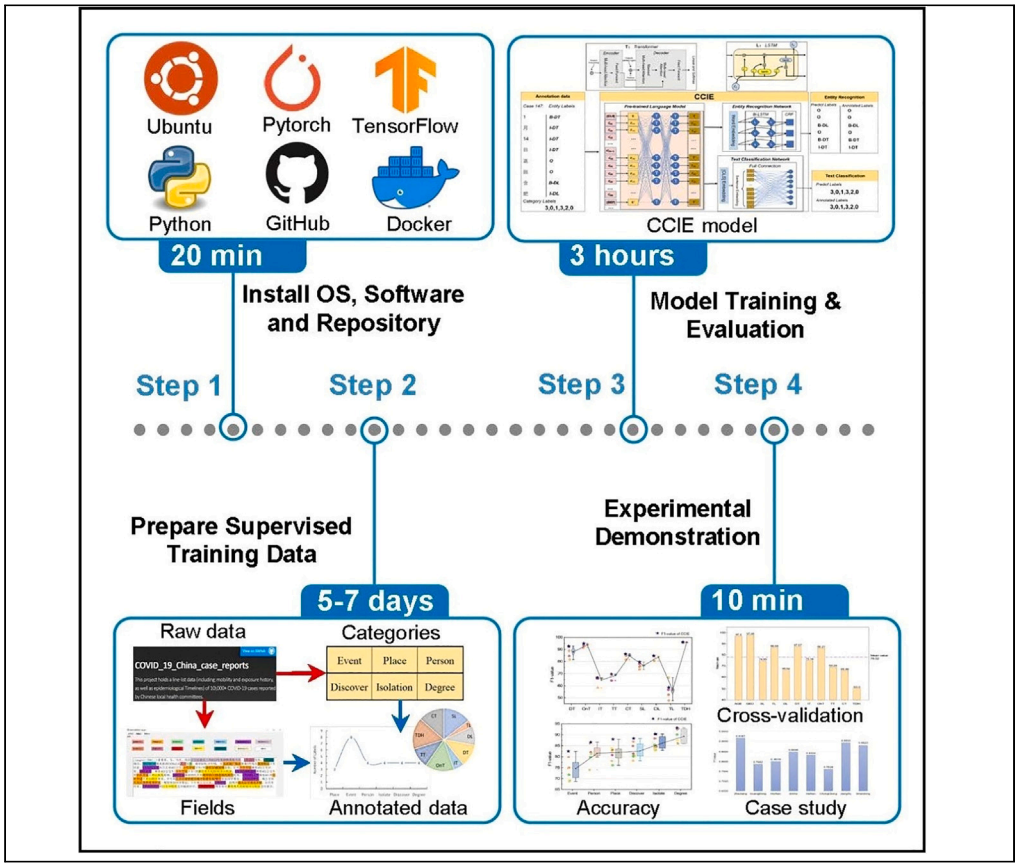
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for the automatic extraction of epidemiological information via a pre-trained language model



The lack of systems to automatically extract epidemiological fields from open-access COVID-19 cases restricts the timeliness of formulating prevention measures. Here we present a protocol for using CCIE, a COVID-19 Cases Information Extraction system based on the pre-trained language model.[1] We describe steps for preparing supervised training data and executing python scripts for named entity recognition and text category classification. We then detail the use of machine evaluation and manual validation to illustrate the effectiveness of CCIE.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Zhizheng Wang, Xiao Fan Liu, Zhanwei Du, ..., Zoie S.Y. Wong, Xiao-Ke Xu, Yuanyuan Sun

zoiesywong@gmail.com (Z.S.Y.W.)
xuxiaoke@foxmail.com (X.-K.X.)
syuan@dlut.edu.cn (Y.S.)

Highlights

Steps for information extraction for COVID-19 case reports using the CCIE approach

Train an information extraction neural network through the executed python code

Details on one-click software installation and epidemiological data preparation

Wang et al., STAR Protocols 4, 102392
September 15, 2023
https://doi.org/10.1016/j.xpro.2023.102392

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for the automatic extraction of epidemiological information via a pre-trained language model

Zhizheng Wang,[1,10] Xiao Fan Liu,[2,10] Zhanwei Du,[3,10] Lin Wang,[4,10] Ye Wu,[5] Petter Holme,[6] Michael Lachmann,[7] Hongfei Lin,[1] Zhuoyue Wang,[1] Yu Cao,[1] Zoie S.Y. Wong,[8,*] Xiao-Ke Xu,[9,*] and Yuanyuan Sun[1,11,12,*]

[1]College of Computer Science and Technology, Dalian University of Technology, 116023, Dalian, Liaoning, China

[2]Web Mining Laboratory, Department of Media and Communication, City University of Hong Kong, Kowloon Tong, Hong Kong Special Administrative Region, 999077, China

[3]WHO Collaborating Centre for Infectious Disease Epidemiology and Control, School of Public Health, Li Ka Shing Faculty of Medicine, The University of Hong Kong, Central And Western District, Hong Kong Special Administrative Region, 999077, China

[4]Department of Genetics, University of Cambridge, Cambridge CB2 3EH, UK

[5]Computational Communication Research Center and School of Journalism and Communication, Beijing Normal University, Beijing 100091, China

[6]Tokyo Tech World Research Hub Initiative (WRHI), Institute of Innovative Research, Tokyo Institute of Technology, Tokyo 152-8550, Japan

[7]Santa Fe Institute, Santa Fe, NM 87507, USA

[8]Graduate School of Public Health, St. Luke's International University, Tokyo 104-0044, Japan

[9]Computational Communication Research Center, Beijing Normal University, Zhuhai, Guangdong, 519087, China; School of Journalism and Communication, Beijing Normal University, Beijing, 100875, China

[10]These authors contributed equally

[11]Technical contact: syuan@dlut.edu.cn

[12]Lead contact

*Correspondence: zoiesywong@gmail.com (Z.S.Y.W.), xuxiaoke@foxmail.com (X.-K.X.), syuan@dlut.edu.cn (Y.S.)
https://doi.org/10.1016/j.xpro.2023.102392

SUMMARY

The lack of systems to automatically extract epidemiological fields from open-access COVID-19 cases restricts the timeliness of formulating prevention measures. Here we present a protocol for using CCIE, a COVID-19 Cases Information Extraction system based on the pre-trained language model.[1] We describe steps for preparing supervised training data and executing python scripts for named entity recognition and text category classification. We then detail the use of machine evaluation and manual validation to illustrate the effectiveness of CCIE. For complete details on the use and execution of this protocol, please refer to Wang et al.[2]

BEFORE YOU BEGIN

This protocol gives a step-by-step guidance to use python code to execute CCIE. We tested CCIE on the Linux system (GPU3090) with the aid of Pytorch and TensorFlow. A different system will likely alter the performances and execution times. Figure 1 shows the overall structure of this protocol, including three steps: data acquisition, data annotation, model training and evaluation.

Installation

Timing: ~20 min (for steps 1 to 3)

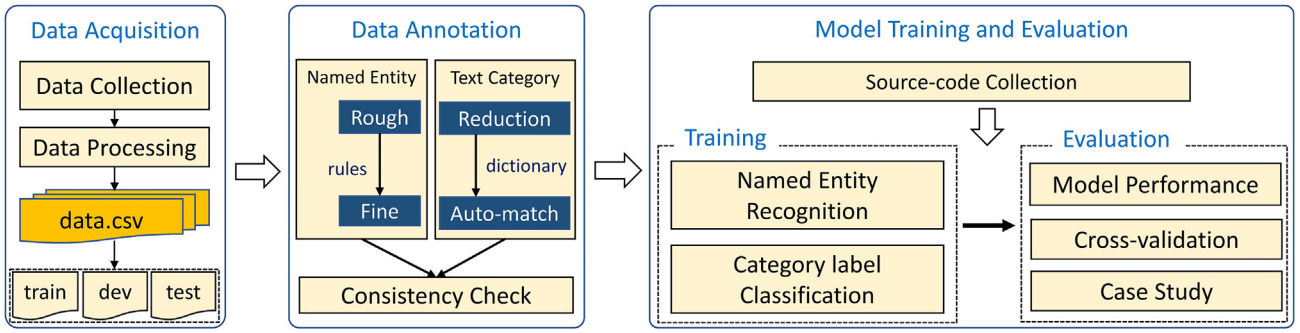


Figure 1. The schematic diagram of this protocol

This step describes the installation scripts for required packages and software.

1. Preparing an Ubuntu operation system (≥ 16.04) unless such an OS is available.

   Alternatives: We provide a docker container (https://hub.docker.com/r/xuce0915/starprotocols) that contains an Ubuntu image. One can use the following command to pull the docker in his/her computer.

```
> docker pull xuce0915/starprotocols
```

2. Installing required software and libraries packages on the Ubuntu terminal. [troubleshooting 1]
   a. Download a Linux-adapted Anaconda from https://www.anaconda.com/products/distribution to install Python 3 (Version ≥ 3.6 is recommended).

```
> bash YOUR_ANACONDA_VERSION.sh
```

   b. Install the TensorFlow (Version ≥1.15 is recommended) from https://www.tensorflow.org/.

```
> pip install tensorflow==1.15.1
```

   c. Install the Pytorch (Version ≥ 1.11 is recommended) from https://pytorch.org/.

```
> pip install torch==1.11.0+cu113 torchvision==0.12.0+cu113 torchaudio==0.11.0 -extra-index-urlhttps://download.pytorch.org/whl/cu113.
```

   d. Install other packages via the pip command.

```
> pip install scikit-learn==0.24.1
> pip install scipy==1.5.4
> pip install pandas==0.25.3
> pip install numpy==1.19.5
> pip install jieba==0.42.1
```

   Note: We have also installed all required software in the *docker*. If one uses our docker, he/she dose not need to install the above software.

Alternatives: We also provide a *requirements.txt* containing all required packages and their versions. One can run the following command to install these packages at one time.

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

3. Preparing the entity annotation tool (In Chinese) by applying to the corresponding author. [troubleshooting 2]

Note: This annotation tool is a plug-and-play application that can be used directly without installation. Its interface is shown in Figure 2, which includes functional components such as file operation, entity annotation options, case report display and annotation progress.

⚠ CRITICAL: Applicants must state the relevant basic information as well as the reasons and necessity of the application. They must also strictly abide by the usage norms and ethical constraints. The entity annotation tool will be provided to applicants after full evaluation within a week. If the applicants are authorized, they should put the tool to the "*CCIE/annotation/*" folder.

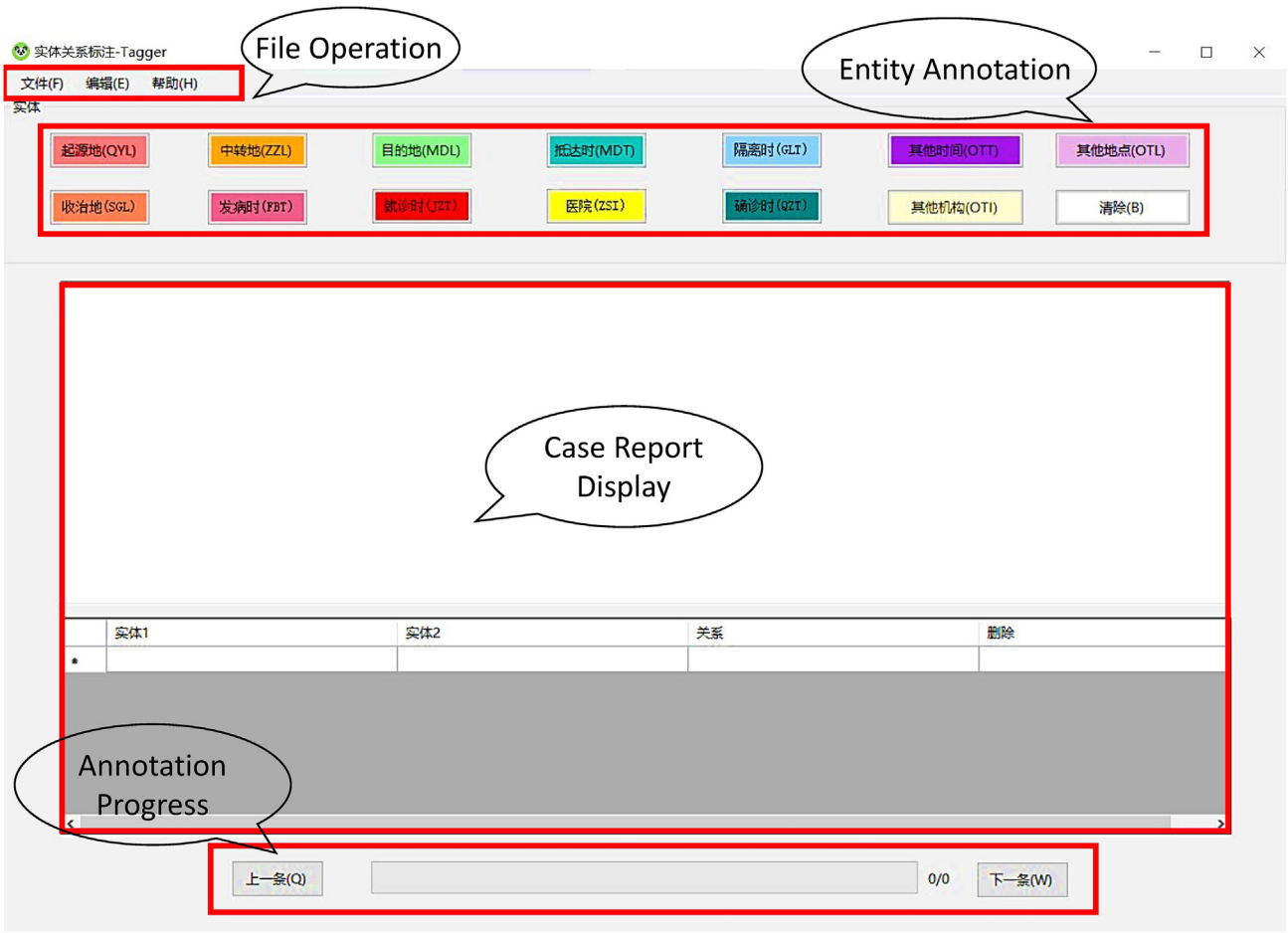


Figure 2. The interface of entity annotation tool

The file operation area contains file import and file exports. The entity annotation area contains all entity labels. The case report display area is used to show the COVID-19 case text. The annotation progress demonstrates the number of case reports that have been annotated.

Alternatives: One can also annotate the raw data according to the step 6.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Raw data and source code	Present study or Liu et al. (2021)[3]	https://github.com/AllenWangle/STARProtocols-CCIE or https://abcdefg3381.github.io/COVID_19_China_case_reports/blob/main/CCIE.zip or Zenodo: https://doi.org/10.5281/zenodo.7941216
Software and algorithms		
Ubuntu	OS ≥ 16.04 LTS	https://hub.docker.com/r/xuce0915/starprotocols
Python	Python v3.6	https://www.python.org/
TensorFlow	TensorFlow v1.15.1	https://www.tensorflow.org/
Pytorch	Torch v1.11	https://pytorch.org/
Numpy	Numpy v1.19.5	https://numpy.org/
Pandas	Pandas v0.25.3	https://pandas.pydata.org/
Scikit-learn	Scikit-learn v0.24.1	https://scikit-learn.org/stable/
Scipy	Scipy v1.5.4	https://scipy.org/
Jieba	Jieba v0.42.1	https://pypi.org/project/jieba/
SPSS	N/A	https://www.ibm.com/spss/
Lattice	Zhang et al. (2018)[4]	https://github.com/jiesutd/LatticeLSTM
TENER	Yan et al. (2019)[5]	https://github.com/fastnlp/TENER
GraphNER	Sui et al. (2019)[6]	https://github.com/DianboWork/Graph4CNER
FLAT	Li et al. (2020)[7]	https://github.com/LeeSureman/Flat-Lattice-Transformer
Chinese Text Classification	Hu (2023)[8]	https://github.com/649453932/Chinese-Text-Classification-Pytorch
Pretrained language model	Google Drive	https://drive.google.com/file/d/1buMLEjdtrXE2c4G1rpsNGWEx7lUQ0RHi/view?
Other		
CPU	Intel(R)	i9-10940X @ 3.30Hz
RAM	N/A	128GB
GPU	NVIDIA	GeForce RTX 3090

STEP-BY-STEP METHOD DETAILS

Here we describe the step-by-step methods for data acquisition and processing, data annotation and consistency check, and model training and evaluation.

Data acquisition and processing

Timing: ~1 h (for steps 1 to 5)

This step illustrates the progress of raw data acquisition and processing, which is used to satisfy the formatting requirements in data annotation.

1. Downloading the GitHub project from https://github.com/AllenWangle/STARProtocols-CCIE and unzipping the file to obtain the raw data (i.e., *dataset_CN.xls*) from the "*CCIE/rawdata*" folder.

   Note: A brief data demonstration is shown in Table 1.

   Alternatives: One also can request raw data from the corresponding author.

2. Extracting the contents of "ID" and "Original_Text_CN" in the *dataset_CN.xls*.

Table 1. An example of raw data

ID	Virus type	Age	…	Original_Text_CN	Original_Text_EN
Anhui_Anqing_1	NA	49		Text in Chinese	Text in English
……	……	……		……	……

3. Splitting the raw data into *train.txt*, *dev.txt* and *test.txt* with a ratio of 8:1:1.
4. Reorganizing each case report in any file into the three-column format.

Note: As shown in Table 2, each line consists of a word, an initially named entity label "O", and a position label "O".

⚠ CRITICAL: The Python code for the step 2, 3, and 4 is provided in the "*CCIE/rawdata/raw_data_process_for_ner_annotation.py*", and one can process the raw data with using the following command.

```
> python raw_data_process_for_ner_annotation.py
```

5. Placing the processed raw data in the "*CCIE/annotation/data*" to be imported into the entity annotation tool for annotation.

Data annotation and consistency check

Timing: ~7 days (for steps 6 to 7)

This step is used for data annotation to prepare supervisory training data.

6. Performing the named entity annotation via the entity annotation tool.

Note: As Figure 3 shows, the named entity annotation contains four key steps, i.e., (a) rough annotation (*step a to c*), (b) fine annotation (*step d to g*), (c) annotation result export (*step h*) and (d) format conversion for model training (*step i*).

a. Start the annotation tool (generally it has been in the "*CCIE/annotation/*" folder).
b. Select the "Open" button in "File" option to import the prepared annotation file (i.e., *train.txt*, *dev.txt* and *test.txt*) from the "*CCIE/annotation/data*" folder.
c. Match human-coded epidemiological fields in 10,017 COVID-19 case reports to the processed raw data, as shown in Figure 3A.
d. Sample 10% of the case reports for manually fine annotation according to the entity distribution obtained by the rough annotation.

⚠ CRITICAL: The sampling objective is to minimize.

$$Loss = \frac{1}{L}\sum_{i=1}^{L}\left(N_{gold}^{i} - N_{sample}^{i}\right) \quad \text{(Equation 1)}$$

where L is the total number of sampled fields, N_{gold}^{i} and N_{sample}^{i} are the number of the i-th label in the manually coded data and the number of the i-th label in the sampled data, respectively.

e. Formulate the rules for annotating named entity annotation.

Note: We select 11 named entities (i.e., L = 11) as the identification targets. They are "age (AGE)", "gender (GED)", "place of departure (SL)", "place of transit (TL)", "arrival date (DT)", "date of quarantine (IT)", "date of symptom onset (OnT)", "date of hospitalization

Table 2. The three-column format for organizing raw data

Cur_List	NER_List	POS_List
ID_1	O	O
#	O	O
Male	O	O
……	……	……

(TT)", "admitted hospital (TDH)", and "date of confirmation (CT)". For each entity, we design the corresponding label symbol (i.e., the abbreviation in brackets) and the annotation span. For example, the span of "Place" is constrained to "City" and the span of "Date" is constrained to "day". More details of annotation rules can be referred to in Appendix Table S1.

f. Annotators require labeling the case reports again according to the established entity annotation rules as shown in Figure 3B.
g. Carry out the consistency check among case reports annotated by different annotators.

Note: To guarantee the consistency and accuracy of the manual annotation, we randomly examined and modified a subset of 100 case reports after they had been annotated by different graduate students. Then, three public-health experts participated in the revisions. After these, we discussed the annotations of the case reports to reach a consensus on the modifications. Next, we continued to examine and manually annotate the remaining case reports. The agreement rate for our revisions reaches 90%, which suggests that the inter-annotator agreement rate is acceptable. Any inconsistent revisions were submitted to the experts for final revisions.

h. Export the annotated results, i.e., the *.xml* file as shown in Figure 3C, to the "*CCIE/annotation/xml*" folder by clicking the "Save as" button in the "File" option.
i. Convert data format from the "xml" style to the "BIO" style for model training.

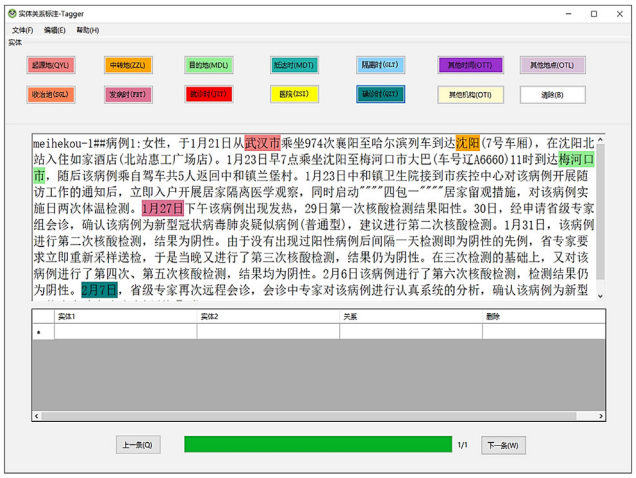
Note: According to the annotated results, the "BIO" labels are employed to prepare the training data. For the token at the start position of an entity, we mark it as "B-M"; For the token at the middle and end position of an entity, we mark it as "I-M". The character "M" denotes the type of entities. For example, as shown in Figure 3D, the admitted hospital "Zhonghe Town Health Center" is annotated as "B-TDH, I-TDH, I-TDH, I-TDH", the date of symptom onset "January 27" is annotated as "B-OnT, I-OnT". For other tokens that beyond the scope of the eleven entities, they are marked with the "O". The script in the "*CCIE/annotation*" folder can be used to obtain the final training data.

```
> python data_postprocess_from_xml.py
```

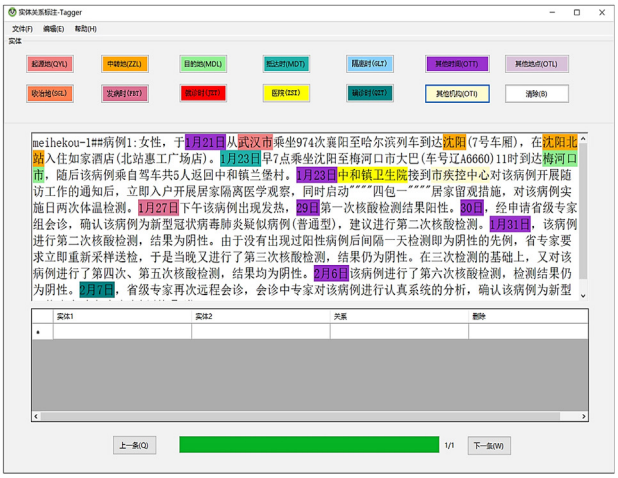
⚠ CRITICAL: We have provided the annotated data in the "*CCIE/ner/datasets/raw/NER*". If one wants to use private data to train the CCIE model, he/she needs to organize an annotation team (about 10 people) to carry out the data annotation according to this step.

7. Performing the category label annotation via the text matching technology.
   a. Integrate the human-coded epidemiological categories involving similar specific scenes into one label.

Note: For example, "hold a party" and "attend a wedding" is considered two different category labels in the epidemiological coding rules, but they are regarded as a "party" label in our annotation rules. In this way, we construct six categories. They are respectively "Place (include



A Rough annotation



B Fine annotation

C Annotation result export

.....	1	月	2	1	日	从	武	汉	市	出	发
O	B-OTT	I-OTT	I-OTT	I-OTT	I-OTT	O	B-QYL	I-QYL	I-QYL	O	O
.....	经	沈	阳	北	站	.....	梅	河	口	市	.....
O	O	B-ZZL	I-ZZL	I-ZZL	I-ZZL	O	B-MDL	I-MDL	I-MDL	I-MDL	O
1	月	2	3	日	中	和	镇	卫	生	院	.....
B-GLT	I-GLT	I-GLT	I-GLT	I-GLT	B-TDH	I-TDH	I-TDH	I-TDH	I-TDH	I-TDH	O
1	月	2	7	日	下	午	.....	2	月	7	日
B-FBT	I-FBT	I-FBT	I-FBT	I-FBT	I-FBT	I-FBT	O	B-QZT	I-QZT	I-QZT	I-QZT

D Format conversion

Figure 3. The process of named entity annotation via the entity annotation tool

(A) The illustration of rough annotation for a COVID-19 case report. Only manually coded fields are marked.

(B) The details of fine annotation that more entities are labeled according to the annotation rules.

(C) The contents of annotated file exported from the entity annotation tool.

(D) The training data format obtained by the annotated file.

three labels)", "Event (include eight labels)", "Person (include four labels)", "Isolate (include four labels)", "Discover (include four labels)" and "Degree (include four labels)". Each of them corresponds to the integrated coding rules. More details can be referred to Appendix Table S2.

b. Derive a vocabulary (named *key_vocab*) to save all category labels and all scenes in a label.

Note: This vocabulary is a data structure in dictionary (i.e., key-value pairs), whose keys correspond to labels and values correspond to key words of scenes. Take the "Place_social" label in the "Place" category as an instance, it is saved as *{Place: {Place_social: {Family, Social place, indoor, School, Work}}}* in the *key_vocab*.

c. Match each report with all the scenes in the *key_vocab* by text matching to determine the labels of case reports.

⚠ CRITICAL: The Python code for the step 7 is provided in the "*CCIE/rawdata/raw_data_process_for_classification_annotation.py*", and one can process the raw data with using the following command.

```
> python raw_data_process_for_classification_annotation.py
```

d. Carry out the consistency check for the matched category labels.

Note: To guarantee the accuracy of the automatic annotation, we randomly examined 100 case reports from all 10,017 case reports. This examination was conducted by three annotators until they fully agreed with the annotation results.

e. Split the data into "train.txt", "dev.txt" and "test.txt" with a ratio of 8:1:1.

⚠ CRITICAL: We have provided the fully processed data in the "*CCIE/classification/data/disease/X*" folder, where "X" denotes a category name.

Model training and evaluation

Timing: ~3 h (for steps 8 to 11)

In this step, we give the detailed scripts of model training and evaluation.

Source code collection

8. Downloading the source code through the following commands:

```
> git clonehttps://github.com/AllenWangle/STARProtocols-CCIE.
> unzip CCIE.zip
```

⚠ CRITICAL: If one wants to use his/her own annotated data, he/she need to put the named entity recognition data (i.e., *train.txt*, *valid.txt* and *test.txt*) to the "*CCIE/ner/datasets/raw/NER*" folder, and put the category classification data (i.e., *train.txt*, *dev.txt* and *test.txt*) to the "*CCIE/classification/data/disease/X*" folder, where "X" denotes a category name.

9. Downloading the pre-trained language model from https://drive.google.com/file/d/1buMLEjdtrXE2c4G1rpsNGWEx7IUQ0RHi/view? and unzip the model to the "*CCIE/ner*" folder and the "*CCIE/classification/*" folder respectively.

Named entity recognition model

10. Training and evaluating the named entity recognition model in the Ubuntu terminal. [troubleshooting 3] [troubleshooting 4] [troubleshooting 5]

```
> cd CCIE/ner

> python train_base_model_bert.py –train True # for training
```

The main parameters are shown as follows:

i. –bert_model_path: chinese_roberta_wwm_ext_L-12_H-768_A-12.
ii. –mode: 0.
iii. –task_name: ner.
iv. –train: True is for training and False is for evaluation.
v. –train_ratio: 1.0.
vi. –language: wzz1.
vii. –emb_drop_rate: 0.2.
viii. –batch_size: 64.
ix. –max_seq_len: 200.
x. –minimal_lr: 1e-4.
xi. –epochs: 40.

Note: The parameters are set as default values in the *train_base_model_bert.py* if one does not set them in the commands. The evaluation metric is F1-score, which is described in Quantification and statistical analysis.

Alternatives: If one uses our docker container, they need to type the following commands to train the named entity recognition model.

```
> docker run --rm -it -v YOUR_LOCAL_PATH_OF_CCIE/ner:/work/CCIE-ner xuce0915/starprotocols ''bash''

> cd work/CCIE-ner

> /root/miniconda3/bin/python train_base_model_bert.py –train True
```

Optional: The function call process after starting the script of model training as follows. One does not need to run the following functions as they are automatically called by the Python script.

a. Call the *process_base_bert*() function to preprocess the training data and generate the input features, which is located in the "*CCIE/ner/utils/prepro_data_lner_BERT.py*". All parameters in this function are defined in the "CCIE/ner/utils/ configs_bert.py".
   i. Configure the BERT language model via the *from_json_file*(*config.bert_config*) function, which is located in the "*CCIE/ner/modeling.py*". The *bert_config* is the configuration file of BERT.
   ii. Read and process the train/valid/test data from the dataset files via *read_data_and_vocab*(*config.train_file, config.dev_file, config.test_file*) function. The *train_file* is the file path of the training data and other two are similar meanings.
   iii. Segmentate sentences to tokens and convert them into numeric numbers via the *FullTokenizer*(*vocab_file = config.vocab_file, do_lower_case = config.word_lowercase*) function, which is located in the "CCIE/ner/tokenization.py". The *vocab_file* is the path of vocabulary file and the *do_lower_case* is the controller for using low case.
   iv. Generate the input features for the training data, validation data and test data via the *file_based_convert_examples_to_features*(*samples, label_list, max_seq_len, tokenizer, output_file*) function, which is located in the "*CCIE/ner/run_classifier_roberta_wwm_large.py*". The *samples* is the input file, *label_list* is the list of label types, *max_seq_len* is the maximum length of the input sequence, *tokenizer* is the segmentation results of step iii, *output_file* is the file path of output results.

b. Call the *_build_model*() function in the *BaseModel_for_BERT*() class to build the named entity recognition model, which is located in the "CCIE/ner/models/base.py".
   i. Load the pre-trained language model via the *bert_embedding*(*input_ids*, *input_mask*, *segment_ids*) function. The *input_ids*, *input_mask* and *segment_ids* are all used to match the parameters of BERT model and are obtained by the *_add_placeholders*() function.
   ii. Shield some neurons randomly with a given probability via the *Dropout(emb_drop_rate)* layer in TensorFlow. The *emb_drop_rate* is the dropout rate of neurons.
   iii. Build the bidirectional LSTM network via the *BiRNN*(*num_units*, *drop_rate*, *concat*, *activation*, *scope = "bi_rnn"*) function to learn long-distance dependency information among different entities. The *num_units* is the number of neural units in LSTM, *drop_rate* is the rate of dropout, *concat* is the concat mode between two layers, *activation* is the activation function. This function is located in the "CCIE/ner/models/shares.py".
   iv. Build the CRF (Conditional Random Fields) layer to learn the possibility distribution of entity label paths via *CRF*(*num_labels*) function. The *num_labels* is the number of entity labels. This function is located in the "CCIE/ner/models/shares.py".
c. Call the *train*() function to train the named entity recognition model, which is located in the "CCIE/ner/models/base.py".
d. Call the *evaluate_data*() function to evaluate the named entity recognition model, which is located in the "CCIE/ner/models/base.py".

⚠ CRITICAL: If one wants to recognize the entities in the new COVID-19 case reports after training the model once, he/she can follow the steps below for data preparation and entity extraction.

a. Type into the "CCIE/ner" folder and put the new case reports in the "Original_Text_CN" column of the "*NewCases.xlxs*". Each line merely contains one case report.
b. Find the *NewCaseExtraction.ipynb* and run the cell under "Pre-process the New Case Reports" to obtain the test data with the same format as the training data.

Note: The original test data in the "CCIE/ner/datasets/NER/test.txt" will be covered. Nevertheless, one can copy the original test data to other folders to prevent this problem.

c. Run the cell under "Evaluation Script for Extracting New Entities". The BIO labels for entities are saved in the "*ckpt/wzz/base_model_wzz1_1_Roberta_wwm_wzz_1_0918/test_result.txt*".
d. Run the cell under "Post-process the Entities in BIO to Readable Entities" to obtain the entities described in the natural language.

Category classification model

11. Typing to the *classification* directory of CCIE, and then using the following command to train and evaluate the category classification model in the Ubuntu terminal. [troubleshooting 3] [troubleshooting 4] [troubleshooting 5]

```
> bash run.sh
```

The main parameters are shown as follows:
   i. –model_name_or_path: chinese_roberta_wwm_ext_pytorch.
   ii. –task_name: disease.
   iii. –do_train: True is for training.
   iv. –do_eval: True is for evaluation.
   v. –data_dir: data/disease/event # *example for the event category*.

vi. –max_seq_length: 512.
vii. –per_gpu_eval_batch_size: 8.
viii. –per_gpu_train_batch_size: 8.
ix. –learning_rate: 4e-5.
x. –num_train_epochs: 3.
xi. –output_dir: output/wzz2_wwm_new.
xii. –model_type: bert.
xiii. –class_type: base.
xiv. –gradient_accumulation_steps: 2.

Note: The parameters are set as default values in *run_glue.py* if one does not set them in the commands. The evaluation metric is also the F1-score.

Alternatives: If one uses our docker container, he/she needs to type the following commands to train the category classification model. [troubleshooting 6]

```
> docker run --rm -it -v YOUR_LOCAL_PATH_OF_CCIE/classification:/work/CCIE-cls xuce0915/starprotocols ''bash''

> cd work/CCIE-cls

> bash run.sh
```

Optional: The function call process after starting the script of model training as follows. One does not need to run the following functions as they are automatically called by the Python script.

a. Load the pre-trained language model and tokenizer via the *from_pretrained*() function.
b. Load the training dataset from the "CCIE/classification/data/disease/X" via the *load_and_cache_examples*(*args.task_name, tokenizer, evaluate = False*) function. The *task_name* is the parameter to point out the datasets, and here is set to *disease*. The *tokenizer* is loaded by the step a. *evaluate = False* denotes there is no validation during the model training.
c. Call the *train*(*train_dataset, model, tokenizer*) function to train the category label classification model. The *train_dataset* is obtained by the step b, and the *model* and *tokenizer* are obtained by the step a.
d. Call the *evaluate*(*model, tokenizer, prefix = global_step*) for model evaluation.

⚠ CRITICAL: If one wants to classify the categories for the new COVID-19 case reports after training the model once, they can remove the *–do_train* term in the "*CCIE/classification/run.sh*", and run the bash run.sh script again.

EXPECTED OUTCOMES

For the input scripts of the named entity recognition task and the category classification task, the expected outputs on the Linux terminal are shown Figures 4 and 5.

Performance evaluation

Named entity recognition

We compared CCIE with four classic deep-neural-network models (i.e., Lattice,[4] TENER,[5] GraphNER,[6] and FLAT[7]) with regard to the recognition of nine entities (see Appendix Table S1). The expected F1-values obtained by the CCIE for all nine entities are shown in Figure 6A. The CCIE demonstrate better performance for most entity recognitions (7/9 cases), including all "*dates*" and two "*places*".

```
Epoch 2/40:
16/16 [==============================] 3.2s/step - Global Step: 32 - Train Loss: 58.1324
dev dataset -- acc: 88.92, pre: 46.02, rec: 57.70, FB1: 51.20
test dataset -- acc: 89.56, pre: 46.60, rec: 57.63, FB1: 51.53
WARNING:tensorflow:From /root/miniconda3/lib/python3.7/site-packages/tensorflow_core/pytho
eprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
From /root/miniconda3/lib/python3.7/site-packages/tensorflow_core/python/training/saver.py
be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
 -- new BEST score on test dataset: 51.53
processed 10626 tokens with 701 phrases; found: 867 phrases; correct: 404.

accuracy:  89.37%; precision:  46.60%; recall:  57.63%; FB1:  51.53

              FBT: precision:  38.67%; recall:  48.33%; FB1:  42.96  75

              GLT: precision:   0.00%; recall:   0.00%; FB1:   0.00  6

              JZT: precision:  14.81%; recall:  16.67%; FB1:  15.69  54

              MDL: precision:  52.78%; recall:  45.24%; FB1:  48.72  36

              MDT: precision:  55.00%; recall:  44.90%; FB1:  49.44  40

              OTI: precision:  10.00%; recall:   6.67%; FB1:   8.00  10

              OTL: precision:  52.26%; recall:  76.47%; FB1:  62.09  199

              OTT: precision:  36.57%; recall:  60.49%; FB1:  45.58  134

              QYL: precision:  41.03%; recall:  66.67%; FB1:  50.79  78

              QZT: precision:  44.09%; recall:  82.00%; FB1:  57.34  93

              SGL: precision:   0.00%; recall:   0.00%; FB1:   0.00  0

              ZSI: precision:  76.47%; recall:  93.81%; FB1:  84.26  119

              ZZL: precision:  34.78%; recall:  14.29%; FB1:  20.25  23
```

Figure 4. The expected outcomes after training the named entity recognition model on the Linux terminal

Category label classification

We compared CCIE with seven benchmark text-classification algorithms (i.e., Transformer,[9] DPCNN,[10] FastText,[11] TextCNN,[12] TextRNN,[13] TextRCNN,[14] and LSTM[15]) for the classification of six categories (see Appendix Table S2). The expected F1-values obtained by the CCIE for all six categories are shown in Figure 6B, with the highest value reaching 93.2%. This result shows that the pre-trained language models obtained word embeddings with richer semantic expression for mining deep features in the text, such as syntactic dependence and semantic role.

Validation between machine extraction and human coding

We compared the 11 machine-extracted fields with the manually extracted ones from the dataset of Liu et al.[3] for the first 10,000 case disclosure reports (i.e., from January 2 to March 4 2020). We used a simple fuzzy matching logic to deal with the style differences between the machine- and human-encoded fields—if the machine-extracted text was present in the manually coded fields, we considered the machine to have provided meaningful information. The expected results are shown in Figure 7. The agreement rates for ages and genders are 97.2% and 97.96%, respectively; those for the places of departure, transit, and destination are 74.95%, 86.84%, and 66.62%, respectively; and those for the dates of arrival, quarantine, symptom onset, hospitalization, and confirmation are 87.67%, 75.14%, 86.21%, 69.24%, and 65.89%, respectively.

Case study

We select eight provinces that reported the highest number of COVID-19 cases (i.e., Zhejiang, Jiangsu, Shandong, Guangdong, Chongqing, Hunan, Anhui, and Henan) and compare the CCIE performance for the reports released by each province. The expected results are shown in Figure 8.

```
Precision, Recall and F1-Score...
              precision    recall  f1-score   support

           0     0.9271    0.9271    0.9271      1015
           1     0.7516    0.7940    0.7722       301
           2     0.5714    0.6667    0.6154        30
           3     0.8632    0.7891    0.8245       256

    accuracy                         0.8752      1602
   macro avg     0.7783    0.7942    0.7848      1602
weighted avg     0.8773    0.8752    0.8758      1602

[[941  52   5  17]
 [ 46 239   2  14]
 [  7   2  20   1]
 [ 21  25   8 202]]
0.8751560549313359
04/17/2023 08:09:38 - INFO - __main__ -   ***** Eval results  *****
04/17/2023 08:09:38 - INFO - __main__ -     acc = 0.8751560549313359
```

Figure 5. The expected outcomes after training the category classification model on the Linux terminal

The CCIE framework perform well for the reports released by the health departments of Zhejiang (91.67%), Jiangsu (89.33%), and Shandong (88.23%) but not for those released by the health departments of Guangdong (78.82%) and Chongqing (76.28%).

QUANTIFICATION AND STATISTICAL ANALYSIS

The t-test was performed using the SPSS tool.

The evaluation F1-value is calculated by the Equation 2

$$P = \frac{TP}{TP+FP}; R = \frac{TP}{TP+FN}; F_1 = \frac{2 \cdot P \cdot R}{P+R} \quad \text{(Equation 2)}$$

where *TP* indicates the number of correct predictions of positive samples, *FP* indicates the number of incorrect predictions of positive samples, and *FN* indicates the number of incorrect predictions of negative samples.

LIMITATIONS

We implement automatic extraction of information from COVID-19 case reports using natural language processing techniques. Such a technical solution currently cannot fully identify specialized information that requires background knowledge and reasoning. Moreover, the performance of CCIE under different operation system and hardware configurations is still unclear.

TROUBLESHOOTING

Problem 1

Related to "step 2 in installation". The versions of software libraries such as Pytorch and TensorFlow will be upgraded, and the built-in functions will also change. This may cause the source code to not work.

Potential solution

Create a virtual environment running CCIE code to share hardware resources with the system configuration and pin the version of the software library in this virtual environment. The commands for creating the virtual environment are as follows:

```
> conda create -n CCIE python==3.6.0

> conda activate CCIE # open the environment

> conda deactivate CCIE # close the environment
```

Problem 2

Related to "step 3 in installation". Due to copyright issues, the Entity Annotation Tool may not be publicly available.

Potential solution

You can organize a team of about ten people to complete the data annotation work according to the provided labeling process.

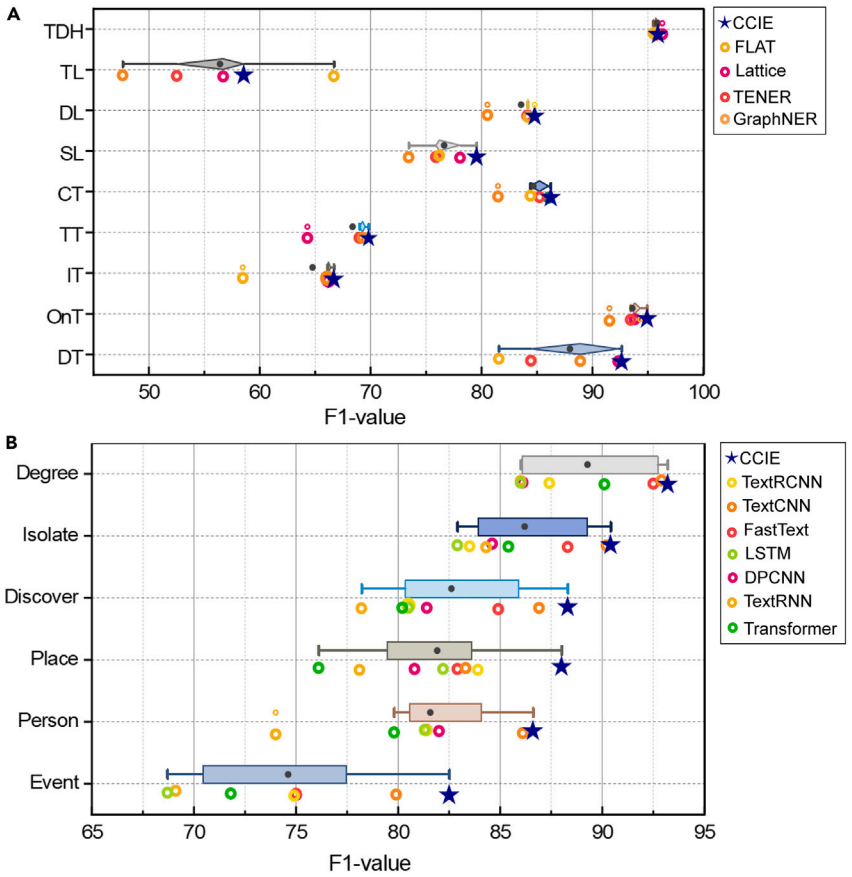


Figure 6. Distributions of F1 values for the named-entity-recognition

We compared CCIE with four classic deep-neural-network and text classification models, obtained using our proposed annotation strategy. This figure was taken from the original publication.[2]

(A) The distribution of the F1 value for each named entity, which aggregates the results of five different named-entity-recognition methods, namely Lattice,[4] TENER,[5] GraphNER,[6] FLAT,[7] and our CCIE (denoted as "★"). The colored distributions correspond to different named entities; (B) the distribution of the F1 value for each text category, which aggregates the results of eight text classification methods, namely Transformer,[9] DPCNN,[10] FastText,[11] TextCNN,[12] TextRNN,[13] TextRCNN,[14] LSTM,[15] and our CCIE. For each named entity or category, the scattered dots indicate the F1 values obtained from different methods, which are used to fit the distribution as indicated by the boxplot.

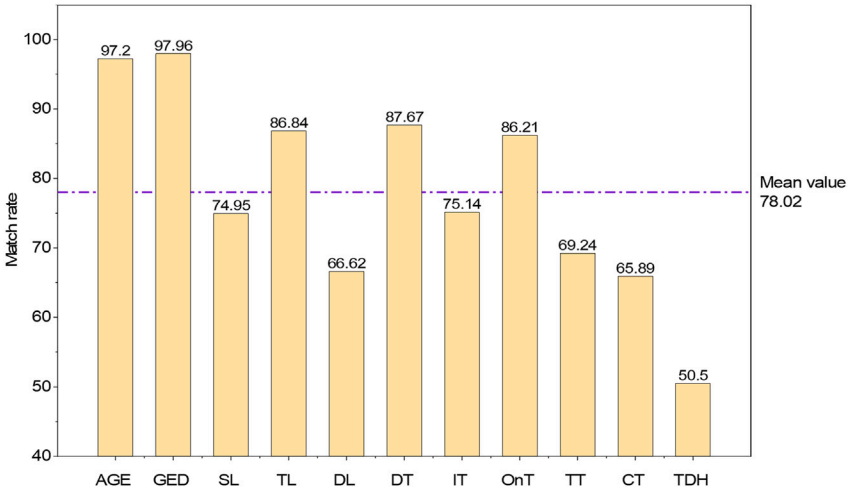


Figure 7. Validation with manually extracted named entities

The accuracy of the eleven data fields identified using our CCIE algorithm as compared to the gold-standard results obtained by manual human coding; the mean value is obtained by averaging the accuracy of all data fields. This figure was taken from the original publication.[2]

Problem 3

Related to "step 10 and step 11 in model training and evaluation". When processing the data or training a neural network, you may encounter that the calculated loss becomes "nan", or the gradient of some parameters becomes "nan" during backpropagation.

Potential solution

You can check your input data. This error will be caused when the input data contains "nan". If you use data in excel format, you need to pay attention to this. When the input data is empty, "nan" may appear during backpropagation.

Problem 4

Related to "step 10 and step 11 in model training and evaluation". Due to the large scale of model parameters, when you are training a neural network, you may face the problem of insufficient GPU memory.

Potential solution

You can use the "nvidia-smi" command to monitor the usage of the GPU to check if there are any unrelated processes occupying memory. In addition, you can reduce the memory usage by reducing the "batch_size" of each epoch. And you can also reduce the number of tensor bits, for example, replace float32 with float16, but this may reduce the accuracy of the model.

Problem 5

Related to "step 10 and step 11 in model training and evaluation". In the process of training the model, you may encounter the problem that the training time is too long, because the pre-trained language model has a large number of parameters.

Potential solution

You can increase the speed of each epoch by increasing the "batch_size", but this may lead to slower model convergence, which can be improved by appropriately increasing the learning rate. If conditions permit, you can use the solid-state drives to increase the IO speed, or pre-load the training data into the memory to reduce IO latency.

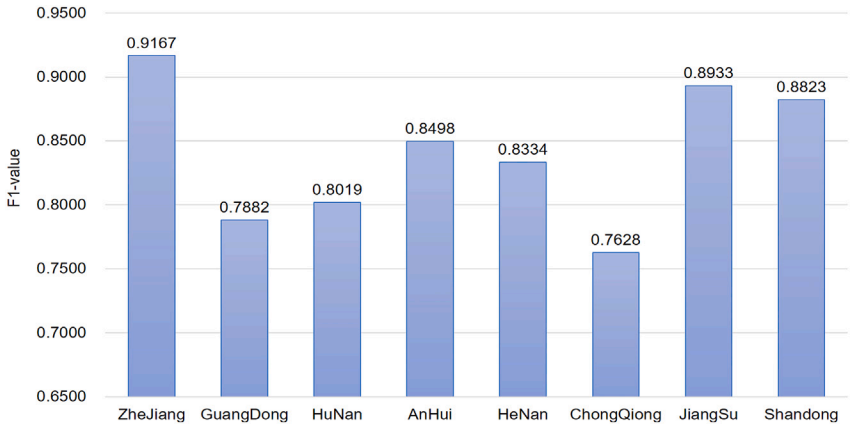


Figure 8. Performance of CCIE in terms of identifying data fields from the COVID-19 case reports disclosed by each province in China

Eight provinces, Zhejiang, Guangdong, Hunan, Anhui, Henan, Chongqing, Jiangsu, and Shandong were selected to assess the effectiveness of the CCIE with regard to the handling of case reports with different natural-language writing styles. This figure was taken from the original publication.[2]

Problem 6

Related to "step 11 in model training and evaluation". When you run python scripts for training the category classification model in docker, there will report errors of "ValueError: Output directory (output/wzz2_wwm_new) already exists and is not empty".

Potential solution

You can remove the "*wzz2_wwm_new*" folder from the "*output*" folder, and then run the python scripts again.

RESOURCE AVAILABILITY

Lead contact

Further information and request should be directed to the lead contact, Yuanyuan Sun (syuan@dlut.edu.cn).

Materials availability

This study did not generate new unique reagents.

Data and code availability

The source code and raw data used in this protocol can be accessed at https://github.com/AllenWangle/STARProtocols-CCIE and are also provided in an open access disposition at Zenodo: https://doi.org/10.5281/zenodo.7941216. Further, one can obtain more data at https://github.com/abcdefg3381/COVID_19_China_case_reports/.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.xpro.2023.102392.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (61773091 and 62173065 to X.-K.X.; 11975025 to L.W.; and 72025405, 82041020, and 91846301 to X.L.); Liaoning Revitalization Talents Program (XLYC1807106 to X.-K.X.); Grand Challenges ICODA pilot initiative, delivered by Health Data Research UK and funded by the Bill and Melinda Gates Foundation and the Minderoo Foundation (to X.F.L.); Japan Society for the Promotion of Science KAKENHI

(Grant No. 18H03336 to Z.S.Y.W.); and the Fundamental Research Funds for the Central Universities (No. DUT22ZD205 to Y.Y.S.).

AUTHOR CONTRIBUTIONS

Methodology, X.-K.X., Y.S.; Investigation, Z.W., X.F.L., Z.D., L.W.; Visualization, Z.W., Z.D.; Funding acquisition, X.-K.X., L.W., X.F.L., Z.S.Y.W., Y.S.; Supervision, Y.W., P.H., M.L., H.L., Z.S.Y.W.; Writing – original draft, Z.W., X.F.L., Z.D., L.W., Z.W., Y.C.; Writing – review & editing, Z.W., X.F.L., Z.D., Z.S.Y.W., X.-K.X., Y.S.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Cui, Y., Che, W., Liu, T., Qin, B., and Yang, Z. (2021). Pre-training with whole word masking for Chinese bert. TASLP, *29*, pp. 3504–3514.
2. Wang, Z., Liu, X.,F., Du, Z., Wang, L., Wu, Y., Holme, P., Lachmann, M., Lin, H., Wong, Z.S.Y., Xu, X.-K., and Sun, Y. (2022). Epidemiologic information discovery from open-access COVID-19 case reports via pretrained language model. iScience *25*, 105079.
3. Liu, X.F., Xu, X.K., and Wu, Y. (2021). Mobility, exposure, and epidemiological timelines of COVID-19 infections in China outside Hubei province. Sci. Data *8*, 54.
4. Zhang, Y., and Yang, J. (2018). Chinese NER using Lattice LSTM. In ACL, pp. 1554–1564.
5. Yan, H., Deng, B., Li, X., and Qiu, X. (2019). Tener: adapting transformer encoder for named entity recognition. Preprint at arXiv. https://doi.org/10.48550/arXiv.1911.04474.
6. Sui, D., Chen, Y., Liu, K., Zhao, J., and Liu, S. (2019). Leverage lexical knowledge for Chinese named entity recognition via collaborative graph network. In EMNLP-IJCNLP, pp. 3821–3831.
7. Li, X., Yan, H., Qiu, X., and Huang, X. (2020). FLAT: Chinese NER using flat-lattice transformer. In ACL, pp. 6836–6842.
8. GitHub, Hu w. (2020). Chinese-Text-Classification-Pytorch. https://github.com/649453932/Chinese-Text-Classification-Pytorch.
9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In NIPS, pp. 5998–6008.
10. Johnson, R., and Zhang, T. (2017). Deep pyramid convolutional neural networks for text categorization. In ACL, pp. 562–570.
11. Joulin, A., Grave, É., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In EACL, pp. 427–431.
12. Kim, Y. (2014). Convolutional neural networks for sentence classification. In EMNLP, pp. 1746–1751.
13. Liu, P., Qiu, X., and Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. In IJCAI, pp. 2873–2879.
14. Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. AAAI, *29*, pp. 2267–2273.
15. Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. (2016). Attention-based bidirectional long short-term memory networks for relation classification. In ACL, pp. 207–212.