Yi Mei · Chao Qian ·
Quan Bai · Bing Xue ·
Sankalp Khanna (Eds.)

LNAI 16452

# PRICAI 2025: Trends in Artificial Intelligence

22nd Pacific Rim International Conference
on Artificial Intelligence, PRICAI 2025
Wellington, New Zealand, November 17–21, 2025
Proceedings, Part II

2 Part II


Springer

MOREMEDIA

Lecture Notes in Computer Science

**Lecture Notes in Artificial Intelligence 16452**

Founding Editor

Jörg Siekmann

Series Editors

# QAACoder: A Question Answering Approach to Actor Detection in the Conflict and Mediation Domain

MohammadSaleh Hosseini[1(✉)], Munawara Saiyara Munia[1], Latifur Khan[1], Patrick Brandt[1], Javier Osorio[2], and Vito D'orazio[3]

[1] The University of Texas at Dallas, Richardson, USA
{seyyedmohammadsaleh.hosseini,munawarasaiyara.munia,lkhan, pbrandt}@utdallas.edu
[2] The University of Arizona, Tucson, USA
josorio1@arizona.edu
[3] West Virginia University, Morgantown, USA
vito.dorazio@mail.wvu.edu

**Abstract.** Monitoring, analyzing, and predicting political turmoil and violence is of utmost importance to a host of political scientists. This is still usually carried out using event coding systems that utilize pattern matching and fixed-size dictionaries. Recently, BERT and ConfliBERT have achieved state-of-the-art results for event coding. However, these methods use a sequence classification paradigm, and thus, are unable to explicitly model the semantics of the labels and the rich interactions among them. In this paper, we propose a novel method for political event extraction on the standard CAMEO (Conflict and Medication Event Observations)-based data set by formulating the problem as question answering overcoming the above drawbacks. We can achieve superior results, improving ConfliBERT, the previous state-of-the-art model, by an absolute F1 of 2.02%. We also propose a new method for multi-source, multi-target sentences that increases the F1 by 2.29% compared to the previous best method.

**Keywords:** Political Actor Detection · Question Answering · CAMEO · BERT · ConfliBERT · Event Coding · Event Extraction

## 1 Introduction

Political event extraction from different news articles is crucial in political science. These extracted data are conventionally utilized by conflict researchers to analyze global interactions between political entities [43]. They also help predict political uncertainty, including domestic and international political disputes, political cooperation, civil conflicts, and violence (e.g., insurgencies, ethno-religious violence, and rebellions), as well as military operations and diplomatic engagements. These predictions and insights can be valuable for policymakers and security sector government agencies in order to effectively address humanitarian and political crises [5].

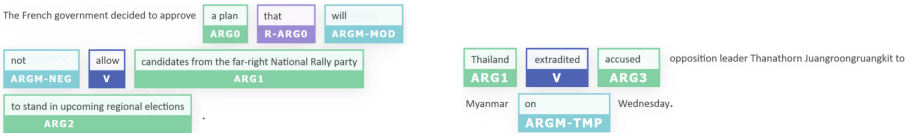

Y. Mei et al. (Eds.): PRICAI 2025, LNAI 16452, pp. 329–345, 2026.
https://doi.org/10.1007/978-981-95-7072-0_23

Automated event-coding systems play an essential role in global violence management by producing data related to political events which aid in modeling and analysis. Systems such as PETRARCH [3], PETRARCH2 [34], and Universal PETRARCH [27] are widely employed by researchers to transform conflict interactions from unstructured text into a structured *who-did-what-to-whom* template, by detecting, extracting, and classifying these interactions from text. These systems leverage pattern-matching techniques and external knowledge bases to perform these tasks. Recently, natural language processing (NLP) techniques have achieved tremendous success in various domains [8,20,21]. In particular, these techniques offer novel prospects for solving some of the key challenges encountered by conventional event-coding methods. Specifically, recent Transformer [48]-based pre-trained language models such as BERT [10], ConfliBERT [22], and Multi-Coped [45] have successfully achieved state-of-the-art results across numerous NLP tasks in the political science domain. While automated event-coding systems have made great progress, accurately extracting political actors (e.g., countries involved in political events) remains a challenge. Before addressing this issue, we first describe the CAMEO ontology and the structure of political event extraction tasks.

A dominant ontology for political event data is **CAMEO** (Conflict and Mediation Domain Event Observations) [16], which features a knowledge base (KB) of political actor dictionaries (containing some 67K entries) and action-pattern dictionaries (about 14K verb phrases). The action-pattern dictionaries contain representations of political interactions, and the actor dictionaries serve as a repository for political entities, such as international entities, national entities, and military non-state actors. CAMEO includes about 200 event types, each corresponding to a set of verbal pattern entries in the action repository. CAMEO functions as a static ontology where this knowledge is stored.

For instance, if we have the following sentence as input to the event coder: "*Sanders and Welch do not support aid to Israel.*", the event coder should output the following: (**WHO**: [Sanders, Welch], **TO-WHOM**: Israel, **DID-WHAT**: 3-Verbal Conflict). **WHO** represents the *sources* of the event, the political actors who initiate(d) the event, and **To-WHOM** represents its *targets*, the actors who are affected in the event. Note that *targets* are not necessarily always the object of the verb related to the event, as in the above-mentioned example, the object of the verb *support* is *aid*, but *aid* is not the target of the event.

The above example shows the intricacies of coding events and one of the reasons why traditional information extraction (IE) [9,12,29] and NLP approaches, such as semantic role labeling (SRL) [13,24,44], cannot address this task. Figure 1 shows two other examples in which a state-of-the-art SRL model (AllenNLP) [15] is incapable of correctly identifying source and targets. The reason is that firstly, the roles sources and targets are not always semantically fixed and vary rested on the event type. As a result, sources and targets are not necessarily associated with *agent* and *theme* in the context of semantic relations, as they can be represented by locations, people, nationalities, organizations, religious groups, political groups, etc.; for instance, location cannot be a *theme* or *agent*).




(a) SRL model detects the target "the far-right National Rally party" correctly, but fails to capture the correct source "The French government".

(b) SRL model correctly detects the source "Thailand", but can not capture the target "Myanmar".

**Fig. 1.** Examples where a state-of-the-art SRL model [15] is not capable of correctly detecting sources and targets.

Second, sources and targets are not always arguments linked to the same verb that ultimately triggers an event. For instance, in the sentence shown in Fig. 1a, SRL properly retrieves the target *candidates from the far-right National Rally party* as one of the arguments for the verb *allow* but does not capture the source *The French Government*. More importantly, there exist instances in which sources and targets are not part of any argument retrieved for any of the triggering verbs (e.g., gold target *Myanmar* in Fig. 1b). Last but not least, for every verb in the sentence, SRL models output its arguments in an input sentence. Choosing which ones are the verbs related to each event type demands an add-on complex module in addition to the SRL model. In order to empirically show that the above-mentioned demerits and challenges render taking advantage of SRL frivolous, we also investigate an SRL model's performance in the results section.

The problem we tackle in this paper is extracting the sources and targets from an input sentence. Following recent work on CAMEO [2,22,45], we assume sentences contain at most one event. Previous methods, such as Multi-Coped and ConfliBERT successfully overcame some of the drawbacks of the above-mentioned methods and set state-of-the-art records on this task. Nevertheless, these methods use sequence classification techniques by trying to assign to each word in the sentence a label such as S (Source), T (Target), and O (Other), which have the following two disadvantages: (1) they cannot explicitly model the semantics of the gold labels, sources and targets, (2) they often fail to capture the rich interactions among them.

Inspired by some prior works [11,25], we formulate the problem of political actor detection as question answering (QA). For instance, by training a system that can answer the question *who is affected in this event?*, given the above sentence, the system can effectively respond with the span *Israel*. In the same manner, by asking the question *Who is the doer of the action in this event?*, the system will find the *sources* (*Sanders* and *Welch*) in the event. Exploiting this technique can solve the above-mentioned demerits by explicitly defining the meaning of *source* and *target*, and the model is able to understand the relation between *source* and *target* better.

Our Contributions in this paper are as follows: (1) We implement a novel approach to political actor detection (2) We propose two new algorithms that improve actor detection accuracy in case of coping with sentences with mul-

tiple sources/targets. (3) We achieve state-of-the-art results for political actor detection on CAMEO.

## 2 Related Work

Researchers interested in the political conflict domain often utilize computational methods to track different conflict events by extracting these events from text in a(n) organized/structured format. While most prior works [3,27,34,36] for event data coding employ pattern-matching techniques using large KBs or domain-specific ontologies, some studies took advantage of classical machine learning and deep learning techniques. Deep neural networks have achieved unprecedented results in different fields of applications [18,28,30–33]. [35] performed event extraction related to prominent political actors from news corpus by exploiting unsupervised learning. [41] proposed a recurrent neural network (RNN) based method for identifying indicators related to protest events. A support vector machine (SVM) based system was utilized by [18] in order to code protest events. [17] introduced a semantic text representations-based method that exploits a joint multilingual semantic vector space. This allowed the authors to use supervised classifiers like SVM and CNN and perform topical coding of sentences from electoral manifestos in languages such as English, German, Italian and French. [37] introduced a logistic regression-based framework to detect Arabic news documents related to conflict and utilized external dictionaries to extract events from text. Finally, ConfliBERT and Multi-Coped utilize BERT to achieve state-of-the-art results on CAMEO source and target detection. ConfliBERT is a domain-specific pre-trained language model tailored for the analysis of political conflict and violence, which holds the state-of-the-art results on CAMEO and many other political tasks. We note that decoder-based large language models, such as GPT [40] or LLaMA [47], generally underperform compared to encoder-based models like BERT or RoBERTa [26] for classification tasks [1,51]. This performance gap is particularly evident in political science applications [6,49]. Moreover, decoder-based models require significantly larger sizes to achieve comparable classification performance, leading to training and inference times that are at least an order of magnitude higher. Due to these limitations, we do not use these models as baselines in our study.

## 3 Model

In this section, we discuss the overview of our model, the question templates that can be used, and how to train and perform inference on the model.

### 3.1 Overview of the Model

An overview of our model is depicted in Fig. 2.
Given a sentence $P = (p_1, ..., p_n)$, our goal is to extract $sources$=$\{(S_{s_1}, S_{e_1}), \ldots, (S_{s_{|sources|}}, S_{e_{|sources|}})\}$ and $targets$=$\{(T_{s_1}, T_{e_1}), \ldots,$

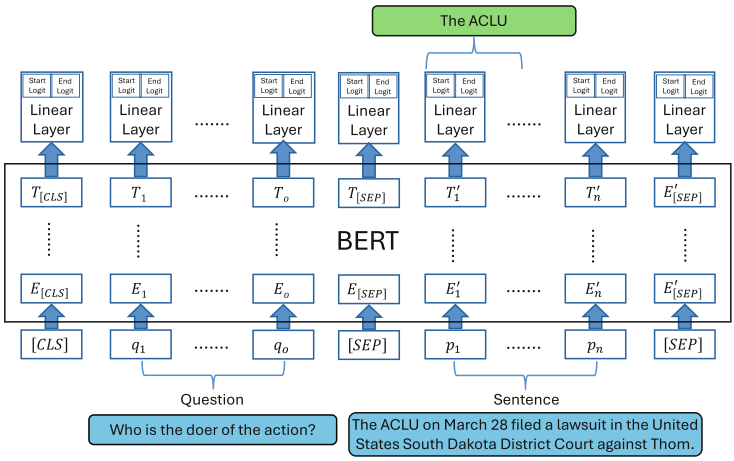



**Fig. 2.** Our proposed question-answering model for solving political actor detection. This question-sentence pair tries to find the *source* of the event.

$(T_{s_{|targets|}}, T_{e_{|targets|}})\}$, where each $S(T)_{s_i}$ and $S(T)_{e_i}$ marks the start and end of the $i$th source (S) or target (T) span ($1 \leq S(T)_{s_i} \leq S(T)_{e_i} \leq n$). We first explain our model for single-source/target sentences (including sentences with no source or target). Multi-source/target sentences (which may contain zero, one, or multiple sources/targets) need a different strategy for training, question formulation, and inference, which we discuss in the next subsection. For single-source/target sentences, we create one question for the source and one for the target. For instance, the question for the source can be "*Who is the source?*", the answer to which is the text span defined in *sources*. We then obtain the tokens for this question $Q = (q_1, \ldots, q_o)$. Then, we use $text = (t_1, \ldots, t_m) = ([CLS], q_1, \ldots, q_o, [SEP], p_1, \ldots, p_n, [SEP])$ as the input to a BERT-like model. BERT outputs vectors $(O_1, \ldots, O_m)$ $(O_i \in \mathbb{R}^h)$, each of which is fed to a linear layer $L : \mathbb{R}^h \rightarrow \mathbb{R}^2$ with weight matrix $W \in \mathbb{R}^{2 \times h}$. We then obtain the log-odds of each token $t_i$ being the start or the end of the answer span by $logits_{start}$ and $logits_{end}$ tuples, respectively, which are defined as follows: $\pi_i(logits_{start}) = \pi_1(L(O_i))$, $\pi_i(logits_{end}) = \pi_2(L(O_i))$ , where $\pi$ is the projection function.

Thus, at training time, $gold\text{-}logits_{start}$ and $gold\text{-}logits_{end}$ are tuples containing all $-\infty$ elements except $\pi_x(gold\text{-}logits_{start})$ and $\pi_y(gold\text{-}logits_{end})$ having values of $\infty$, where $x$ and $y$ mark the start and end of the gold answer span $(x, y)$, respectively. If there is no gold answer, the index of $[CLS]$ is chosen as the start and the end of the answer span. We create a $(Q, P, A)$ triplet for each source and target and for each $P$, where $A$ is the answer span. We can then train the model with these instances.

At inference time, we pass *text* to the trained model and retrieve $logits_{start}$ and $logits_{end}$. Afterwards, we descendingly sort all the possible answer spans

**Table 1.** Question templates

| Question Text for Source | Question Text for Target |
|---|---|
| 1. [Source] | 1. [Target] |
| 2. Who is the source? | 2. Who is the target? |
| 3. What entity is the source? | 3. What entity is the target? |
| 4. **Who is the doer of action in the event?** | 4. **Who is affected in the event?** |

$(s, e)$ based on their $\pi_s(logits_{start}) + \pi_e(logits_{end})$. We then remove the invalid spans from this sorted list, such as spans that fall out of the sentence or have $e > s$. We then select the first element of the remaining list as our answer.

We tried with the question templates in Table 1 for *sources* and *targets*, and we found out that the best results are acquired by choosing Q4 for both source and target. We present the results for the other question templates in Table 4.

### 3.2 Multi-source, Multi-Targets

For multi-source/target sentences, at training time, the same strategy as in the single-source (target) is used; however, here, a separate $(Q, P, A)$ is made per each $A$ (gold answer). The same template is used for making the questions.

The inference in the case of multi-source/target sentences is challenging since the model is trained to predict one single span for the same question-passage pair. We can still use the same strategy for inference by the same sorting method used in the single-source case. However, the question would be how many best spans we should choose as the predictions since the question may have zero, one, or multiple answers. To this end, we can use a dynamic thresholding method as suggested by [11] expressed in Algorithm 1 (when $rmvOverlap = False$ and $useTrainData = False$). This algorithm is applied after each training epoch on the development dataset. First, it descendingly sorts all the spans (alongside their question ID and text) in all the sentences in the development data set based on their probability of occurrence ($flatSpans$). This probability for each span $(s, e)$ in the instance with the question id $qaID$ is equal to $prob_{qaID,s,e} = spanProb_{qaID,s,e} - naProb_{qaID}$, where $spanProb_{qaID,s,e} = \pi_s(logits_{start_{qaID}}) + \pi_e(logits_{end_{qaID}})$ and $naProb_{qaID}$ is the probability that question $qaID$ does not have an answer. The algorithm then starts with an empty set of candidate answer spans ($candidateSpans$). At each step, it adds the next best span on the sorted list $flatSpans$ to $candidateSpans$ and calculates $F1$ on the development data set. If $F1$ is higher than the best $F1$ ($bestF1$), it updates $bestF1$ to $F1$, while recording the current $prob_qaID, s, e$ as the best threshold ($bestThresh$). The algorithm finally returns $bestThresh$.

However, using the above-mentioned algorithm leads to relatively poor results as we show in Table 3; hence, we propose these methods for ameliorating said algorithm:

**Algorithm 1:** Find Best Threshold

```
findThreshold(Array[Array] devSpansList, Array devQaIDs, Array[Array] devGoldSpansList, Bool rmvOverlaps, Bool useTrainData, Array[Array] trainSpansList, Array trainQaIDs, Array[Array] trainGoldSpansList):
if useTrainData then
    allSpansList ← devSpansList + trainSpansList;
    allQaIDs ← devQaIDs + trainQaIDs;
    allGoldSpansList ← devGoldSpansList + trainGoldSpansList;
else
    allSpansList ← devSpansList;
    allQaIDs ← devQaIDs;
    allGoldSpansList ← devGoldSpansList;
foreach spans in allSpansList do
    ;                          // Each span in spans is a tuple = (spanText, spanProb)
    allSpansList [i] ← sort(spans, key=spanProb) ; // Descending sort based on probabilities
if rmvOverlaps then
    foreach spans in allSpansList do
        foreach (spanText, spanProb) in spans do
            foreach (spanText2, spanProb2) in spans do
                if overlap(spanText, spanText2) then
                    delete(spans[k]);
flatSpans ← Array();
foreach spans, qaID in allSpansList, allQaIDs do
    naProb ← findNoAnswerProb(spans);
    foreach (spanText, spanProb) in spans do
        flatSpans.append(qaID, spanText, naProb − spanProb);
flatSpans ← sort(flatSpans, key=naProb − spanProb) ;  // Descending sort based on the third tuple's value
flatGoldSpans ← Array();
foreach spans, qaID in allGoldSpansList, allQaIDs do
    foreach spanText in spans do
        flatGoldSpans.append(qaID, spanText);
bestThresh ← 0;
bestF1 ← 0;
candidateSpans ← Array();
foreach (qaID, spanText, prob) in flatSpans do
    candidateSpans.append(qaID, spanText);
    curF1 ← calcF1(candidateSpans, flatGoldSpans);
    if curF1 > bestF1 then
        bestThresh ← prob;
        bestF1 ← curF1;
return bestThresh;
```

(1) We utilize the training data as well. More data makes the threshold more accurate (lines 2–5 in Algorithm 1 when $useTrainData = True$). (2) We remove all but the best overlapping spans of answers for a particular (Q, P) pair (lines 12–17 in Algorithm 1, when $rmvOverlap = True$). Starting from the first span, we iterate the rest of the spans and remove all the overlapping ones. Then we go to the second item in the new list and perform the same. We do this until no extra span is left. This means that whenever two spans overlap, we eliminate the span having lower log-odds, keeping the other span.

Further, we propose a new method for finding the number of best spans to choose from, which we call **Knee Point Detection**, described in Algorithm 2. The idea behind this algorithm is that logically, correct answer spans generally form a cluster due to their high probabilities, while incorrect spans cluster separately with lower probabilities. The knee point refers to a position where a significant drop in the span probabilities occurs, marking the transition from correct to incorrect spans (Fig. 3 provides a visualization of knee point). This provides an effective stopping criterion for selecting correct spans. Algorithm 2 trains a classifier to differentiate between valid and invalid answer spans. For each $(spans, goldSpans)$ pair in the training data, it sorts the spans by their probability ($spanProb$). Then, it compares the probability of each span to the next one's, calculating the probability difference ($P_{diff} = curSpanProb - nextSpanProb$). If the current span's index corresponds to the number of gold spans (line 10 in the Algorithm 2), it appends $P_{diff}$ to the samples list and assigns it a label of 1 to indicate that knee point occurs here. The break statement is then used to

**Algorithm 2:** Knee point detection

```
1  trainClassifier(Array[Array] trainSpans, Array[Array] trainGoldSpans,
     String clfName):
2  samples ← Array();
3  sampleLabels ← Array();
4  foreach spans, goldSpans in trainSpans, trainGoldSpans do
5      spans ← sort(spans, key=spanProb) ;          // Sort based on spanProb
6      for i ← 0 to len(spans) −1 do
7          (curSpanText, curSpanProb) ← spans[i];
8          (nextSpanText, nextSpanProb) ← spans[i + 1];
9          P_diff ← curSpanProb − nextSpanProb;
10         if len(goldSpans) == i then
11             samples.append(P_diff);
12             sampleLabels.append(1);
13             break break ;                        // Move to the next set of spans
14         else
15             samples.append(P_diff);
16             sampleLabels.append(0);
17 trainedClf ← trainClf(samples, sampleLabels, clfName);
```

exit the loop. If the above condition is not met, the algorithm appends $P_{diff}$ with a label of 0, marking it as being not a knee point. Finally, the algorithm trains the classifier using *samples* and *sampleLabels* and returns it.

After finding the best threshold, we apply it to the list of spans for a particular $(Q, P)$ pair, filtering out all the undesired spans. The remaining spans are chosen as the answers to question $Q$.

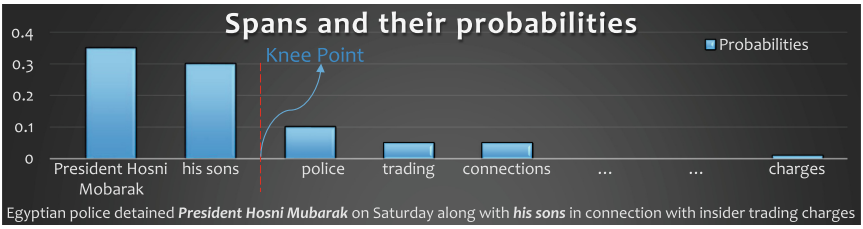



**Fig. 3.** An illustration of where knee point occurs in a sentence based on the sorted possible answer spans for detecting the targets bolded in the sentence.

We can choose any classifier for this algorithm. We chose Random Forest [7] and achieved promising results. Gradient Boosting [14] yielded similar results as well. After we get the trained classifier $trainedCLF$, at test time, to choose N (number of answers to be chosen) for each of the questions: 1. set i = 1. 2. Choose the answer spans i and i+1 returned by the model and calculate their probability difference $P_{diff}$. 3. Pass $P_{diff}$ to the classifier. If $trainedCLF$ returns 1, set $N = i$ and go the next question. Otherwise, set $i = i + 1$ and go to step 2. In this way, we can get the correct number of spans for each of the questions.

## 4 Experiments and Results

In this section, we describe the experimental setup and the data sets we used. Then, we discuss the baselines, and finally, the results we obtained and the discussion on them.

### 4.1 Data Set and Evaluation Setup

For our experiments, we use a CAMEO-based data set consisting of 1002, and 330, 331 training, development (dev), and test samples, respectively. The samples are collected from various worldwide news agencies. The annotations sources, and targets are independently carried out by six bachelor's students specifically trained for this task and revised by professors and graduate students expert in conflict analysis. A Fleiss' kappa of 61% was achieved for inter-annotator agreement on sources and targets, which is considered apt for this task [45].

This data set is composed of samples that have zero, one, or multiple sources and/or targets. We conduct two sets of experiments to show the effectiveness of

our model and methods. The first experiment uses only sentences that have single or zero sources or targets. The second experiment includes all the data points (including multi-source/target sentences). For the single-source/target data set (including sentences with no source/target), we have 881, 294, and 295 data points in the training, dev, and test data sets, respectively.

### 4.2 Baselines

To show the efficacy of our method, we compare our method to a variety of baselines. First, we investigate the widely-used frameworks for CAMEO-coded political event extraction, namely the **PETRARCH** family. These event coders use parse trees and a large fixed dictionary of political actors and verb patterns to capture sources and targets in a sentence. This family includes three different coding systems: **PERTARCH**, **PERTARCH2** and **UPETRARCH**. PETRARCH [3] and PETRARCH2 [34] exploit constituency parse trees for event coding, whereas UPETRARCH [27] uses dependency parse trees to do so.

In addition, we use the same SRL methods used in [45]. please refer to the appendix for more details on these models.

Moreover, we used an LSTM-CRF-based model (**BiLSTM-CRF**) for sequence labeling tasks [19,46], which consists of two bidirectional Long short-term memory (BiLSTM) attached to a Conditional Random Field (CRF). We used 300-dimensional pretrained word embeddings from fastText [4] as input to the initial BiLSTM layer. As discussed in Sect. 2, we do not include decoder-based models (e.g., GPT and LLaMA) as baselines in this study.

We applied our question-answering method to certain backbone BERT-like models. These models attached to a classification layer (as discussed in Sect. 1) are also used as our baselines. These models include BERT-base-cased (**BBC**) (used in Multi-coped [45]), RoBERTa (**RB**), and two versions of ConfliBERT [22]: ConfliBERT$_{scr-uncased}$ (**CBSU**) and ConfliBERT$_{cont-cased}$ (**CBCC**). ConfliBERT achieved state-of-the-art results on CAMEO data set. We tried other versions of ConfliBERT as well, yet consistent with [22], those did not yield superior results to the other two versions (though applying our method improved them as well). We also tried BERT-base-uncased, but we achieved similar results to the cased version.

### 4.3 Experimental Setup

We train our models and the baselines up to 20 epochs, repeated 10 times with 10 different random initializations. The best epoch for each seed is chosen based on the dev data set, and the average test results at those epochs are reported. For all the models, we use a mini-batch size of 16, a maximum sequence length of 128, and Adam optimizer [23] with a learning rate of 10e-5.

As for the performance metrics we use F1 and Micro Precision, Recall, and F1. The F1 is used for individual labels Source and Target, while the Micro metrics are used when the mixed performance of these two labels is considered.

Please refer to the appendix for information on the computer and software packages we used.

### 4.4 Results and Discussion

The results for single-source/target sentences are shown in Table 2. Our models are shown with a **QA** postfix. The paired t-test was performed between each model and its corresponding baseline (e.g., BBC vs. BBC-QA) in order to show the statistical significance of our experiments.

**Table 2.** Exact-match F1 for sentences with single-source/target. Bold values are the best result between a baseline and its corresponding QA model. Stars denote the best performance across all models.

| Model | Source F1 | Target F1 | Micro Prec | Micro Rec | Micro F1 |
|---|---|---|---|---|---|
| PETRARCH | 42.79 | 21.30 | 47.19 | 23.06 | 30.99 |
| PETRARCH2 | 28.34 | 15.77 | 32.69 | 11.40 | 22.05 |
| UPETRARCH | 33.62 | 11.07 | 26.47 | 17.70 | 21.22 |
| SRL-based | 85.07 | 12.41 | 45.34 | 46.29 | 45.81 |
| SRL-based (UB) | 88.53 | 18.24 | 53.14 | 49.19 | 51.09 |
| BiLSTM-CRF | 87.48 | 62.71 | 76.50 | 74.44 | 75.45 |
| BBC | 93.60 | 73.19 | 81.45 | **85.41** | 83.38 |
| BBC-QA | **95.20** | **74.86** | **85.23** | 85.02 | **85.12** |
| RB | 94.40 | 72.80 | 81.88 | **85.35** | 83.58 |
| RB-QA | **95.26*** | **74.70** | **85.08** | **85.08** | **85.08** |
| CBSU | 94.60 | 74.86 | 83.02 | 86.46 | 84.70 |
| CBSU-QA | **94.66** | **78.64*** | **86.89*** | **86.55*** | **86.72*** |

As we spot, our models overall outperform their corresponding baselines by notable margins. For instance, considering the most important metric, which is Micro F1, we are able to improve over BBC, RB, and CBSU with absolute F1 improvements of 1.80%, 1.50% and 2.02%, respectively (with p-values less than 0.001, 0.059, and 0.001, respectively). CBSU-QA also sets a new state-of-the-art record on this data set for single source/target sentences with an improvement of 2.02% over the previous best model, CBSU. We also observe that our models improve both individual Source F1 and Target F1 with decent margins.

### 4.5 Multi-Source/Target Sentences

Table 3 lists the results obtained for all the test samples which include multi-source/target sentences as well. As can be seen, our models improve over their

**Table 3.** Exact-match F1 for multi-source/target sentences. See Table 2 for starred and bold numbers' meanings

| Model | Source F1 | Target F1 | Micro Prec | Micro Rec | Micro F1 |
|---|---|---|---|---|---|
| PETRARCH | 45.99 | 21.05 | 50.37 | 23.99 | 32.50 |
| PETRARCH2 | 29.54 | 15.96 | 35.01 | 10.41 | 22.71 |
| UPETRARCH | 35.55 | 11.63 | 27.78 | 18.75 | 22.39 |
| SRL-based | 84.23 | 13.75 | 46.61 | 45.71 | 46.15 |
| SRL-based (UB) | 87.70 | 19.76 | 54.51 | 48.69 | 51.43 |
| BiLSTM-CRF | 88.60 | 60.56 | 74.96 | 74.47 | 74.72 |
| BBC | 93.32 | 70.05 | 80.25 | 83.33 | 81.76 |
| BBC-QA$_{org}$ | 91.89 | 68.03 | **88.64** | 74.25 | 80.79 |
| BBC-QA$_{alg1}$ | 93.08 | 72.83 | 85.12 | 81.39 | 83.21 |
| BBC-QA$_{kneepoint}$ | **94.20*** | **74.55** | 84.81 | **84.07** | **84.44** |
| RB | 92.80 | 72.39 | 81.10 | **84.14** | 82.59 |
| RB-QA$_{org}$ | 93.22 | 68.88 | **90.39*** | 75.06 | 81.94 |
| RB-QA$_{alg1}$ | 93.12 | 73.33 | 86.54 | 80.87 | 83.58 |
| RB-QA$_{kneepoint}$ | **93.80** | **76.09** | 86.06 | 84.02 | **85.03** |
| CBCC | **93.44** | 72.7 | 82.09 | 84.23 | 83.14 |
| CBCC-QA$_{org}$ | 93.02 | 75.29 | **88.91** | 80.58 | 84.51 |
| CBCC-QA$_{alg1}$ | 93.20 | 76.45 | 87.10 | 83.07 | 85.04 |
| CBCC-QA$_{kneepoint}$ | 93.42 | **77.34*** | 86.37 | **84.52** | **85.43*** |
| CBSU | 93.99 | 72.57 | 81.43 | **85.07*** | 83.21 |
| CBSU-QA$_{org}$ | 93.22 | 73.00 | **90.02** | 78.32 | 83.74 |
| CBSU-QA$_{alg1}$ | 94.05 | 75.39 | 86.85 | 83.17 | 84.97 |
| CBSU-QA$_{kneepoint}$ | **94.07** | **76.18** | 86.16 | 84.26 | **85.20** |

**Table 4.** Performance of different types of questions

| Question template | Source F1 | Target F1 | Micro Prec | Micro Rec | Micro F1 |
|---|---|---|---|---|---|
| 1 | 96.47 | 77.14 | 87.00 | 86.90 | 86.95 |
| 2 | 96.35 | 78.60 | 87.78 | 87.47 | 87.63 |
| 3 | 96.49 | 78.38 | 87.71 | 87.44 | 87.58 |
| 4 | **96.65** | **79.10** | **88.38** | **87.72** | **88.05** |

baselines by promising margins. Our knee point method coupled with Random Forest outperforms all the other correspondent baselines on Micro F1. For example, CBCC-QA and CBSU-QA show absolute Micro F1 improvements of 2.29% and 1.99% over CBCC and CBSU. Hence, we achieve a new state-of-the-art performance on the CAMEO dataset, reaching a Micro F1 of 85.04%, outperforming the previous best system, which has a Micro F1 of 83.14%.

Besides, **Alg1** method in the table, which is Algorithm 1, substantially improves most of the metrics over **org** method (the method used in [11] (expressed in Algorithm 1 ($rmvOverlap = False$ and $useTrainData = False$)).

We also achieve superior Micro Precision performances (improvements of around 3.6%); however, we spot that by applying our model, the recall is slightly hurt (around 1.5%). Nonetheless, CBCC-QA gets the least performance drop (1.09%) which combined with its 1.90% F1 and 3.74% precision improvements, makes it a desirable model to be used for political actor detection on CAMEO.

Moreover, note that compared to the single-source/target case, the performance drop in recall is notably higher when considering all the samples. This is because in the single-source/target case, opposite to the all-data scenario, no threshold is applied and the best answer span is always returned; this results in a strict threshold that filters out as many spans as possible leaving some correct spans left out, hence the lower recall. We leave the solution to alleviate this issue to future work.

### 4.6 Error and Performance Analysis

Our analysis shows the following errors:

- **Capturing an incomplete noun phrase**: In the instance "*Police arrested* $\underline{\textit{a man on a London-bound flight}}_{gold\ target}$ *on suspicion of [...]*", our model partially captures *a man* as the target.
- **Wrong capture**: In the instance, "*Isis fighters have surrendered their final scrap of territory in Syria to* $\underline{\textit{US-backed forces}}_{gold\ target}$*, giving up land [...]*", our model captures both the gold target as well as *Syria* as the targets.

Beside the quantitative results, consider the following sentence as to why the QA method outperforms classification. "*[...] said that* $\underline{\textit{Taiwanese business people}}_{source}$ *based in Indonesia have started a campaign that has collected $80,000 to help* $\underline{\textit{earthquake victims}}_{target}$". Our model correctly identifies the source and target, while the baseline does not find any source and mistakes the true source for the target. This shows that *knowing the meaning* of the labels helps the QA model predict better.

### 4.7 Effect of Different Questions

The results in Table 4 show that the question formulation is important as it helps the model grasp the meaning of the gold labels source and targets as discussed in Sect. 1. This empirically shows why QA performs better than classification.

## 5 Conclusion

Political and social scientists and governmental agencies constantly need to gather and analyze event data on conflict processes around the globe. As a result, researchers rely on computer-generated event data. Presently, most of these event

coding protocols depend on pattern-matching approaches constrained by large dictionaries; however, recently methods rested upon large language models such as ConfliBERT have been able to drastically increase the performance of event coders, achieving state-of-the-art results on CAMEO-based corpora. These methods, nevertheless, use sequence classification for *source*/*target* (political actor) detection, which has the drawback of not being able to explicitly utilize the meaning of the labels *sources* and *targets*, as well as the interaction among them.

In this paper, we propose to formulate the political actor detection problem as question answering. By doing so, we are able to mitigate said drawbacks. We demonstrate the effectiveness of the proposed architecture by testing it on a CAMEO-based corpus. Our results empirically show that leveraging question answering is beneficial for political actor detection. We also introduced two new algorithms that boost the performance of our models when coping with multi-source/target sentences. A possible venue for future work would be to extend this technique to multi-lingual settings, as well as possibly apply this technique to other political science domains.

**Acknowledgments.** This work was partially supported by NSF OAC-2311142.

## A SRL-Based Methods for Event Coding

We experimented a state-of-the-art BERT-based model for SRL implemented in AllenNLP [15]. As explained in Sect. 1, SRL outputs multiple tuples of arguments (semantic roles). Usually, ARG0 and ARG1 correspond to the source and target, respectively. Ergo, we extract sources and targets from these outputs in two ways in which we consider ARG0 as the source and ARG1 as the target. In the first version, called SRL-based, we choose the first tuple having both ARG0 and ARG1. In the second version, called SRL-based (UB), we choose the tuple having ARG0 and ARG1 that maximizes Macro F1 when calculated against the gold sources and targets. Note that SRL-based (UB) works only as an upper-bound baseline since it is unimplementable in practice. It simply shows what would the best possible performance we could reach by using SRL model.

## B Computer and Software Packages Used

To conduct the experiments discussed in this paper, we used a computer with an NVIDIA GeForce RTX 8000 GPU, 128GB of RAM, and an Intel Core i9-9980XE CPU. Our code uses PyTorch [38], huggingface [50], sci-kit learn [39], and Simple Transformers [42].

## References

1. Bahak, H., Taheri, F., Zojaji, Z., Kazemi, A.: Evaluating chatgpt as a question answering system: A comprehensive analysis and comparison with existing models (2023). arXiv: 2312.07592 [cs.CL]

2. Beieler, J.: Generating politically-relevant event data. In: Proc. Workshop on NLP and Computational Social Science, pp. 37–42 (2016)
3. Beieler, J., Norris, C.: Petrarch: Python engine for text resolution and related coding hierarchy (2014)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. ACL **5**, 135–146 (2017)
5. Boschee, E., Lautenschlager, J., O'Brien, S.: Icews coded event data. Harvard Dataverse (2015). https://dataverse.harvard.edu
6. Bosley, M., Jacobs-Harukawa, M., Licht, H., Hoyle, A.: Do we still need bert in the age of gpt? comparing the benefits of domain-adaptation and in-context-learning approaches to using llms for political science research. In: Annu. Meet. of the Midwest Poli. Sci. Assoc. (MPSA) (2023)
7. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
8. Chalkidis, I., Fergadiotis, M., Malakasiotis, P., Aletras, N., Androutsopoulos, I.: LEGAL-BERT: The muppets straight out of law school. In: Findnings of EMNLP, pp. 2898–2904 (Nov 2020)
9. Cui, L., Wei, F., Zhou, M.: Neural open information extraction. In: Proc. of the ACL, pp. 407–413. Melbourne, Australia, July 2018
10. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proc. of NAACL-HLT, pp. 4171–4186. ACL, Minneapolis, Minnesota, June 2019
11. Du, X., Cardie, C.: Event extraction by answering (almost) natural questions. In: Proc. 2020 Conf (EMNLP), pp. 671–683. ACL, November 2020
12. Fader, A., Soderland, S., Etzioni, O.: Identifying relations for open information extraction. In: Proc. of EMNLP, pp. 1535–1545. Edinburgh, Scotland, UK. (Jul 2011)
13. Faruqui, M., Dyer, C.: Improving vector space word representations using multilingual correlation. In: Proc. of European Chapter of ACL, pp. 462–471. Gothenburg, Sweden, April 2014
14. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics, pp. 1189–1232 (2001)
15. Gardner, M., et al.: AllenNLP: a deep semantic natural language processing platform. In: Proc. of workshop for NLP and Open Source Software, pp. 1–6. Melbourne, Australia (Jul 2018)
16. Gerner, D.J., Schrodt, P.A., Yilmaz, O.: CAMEO conflict and mediation event observations event and actor codebook (2012)
17. Glavaš, G., Nanni, F., Ponzetto, S.P.: Cross-lingual classification of topics in political texts. In: Proc. of Workshop on NLP and Comput. Soc Sci., pp. 42–46. Vancouver, Canada, August 2017
18. Hanna, A.: Mpeds: Automating the generation of protest event data, January 2017
19. Hochreiter, S.: Long short-term memory. Neural Computation MIT-Press (1997)
20. Hosseini, M., Munia, M., Khan, L.: BERT has more to offer: BERT layers combination yields better sentence embeddings. In: Findings of EMNLP, pp. 15419–15431. Singapore, December 2023
21. Hosseini, S.M., Sameti, H.: Creating a corpus for automatic punctuation prediction in Persian texts. In: 2017 Iranian Conf. on Electrical Engineering (ICEE), pp. 1537–1542 (2017)
22. Hu, Y., Hosseini, M., Skorupa Parolin, E., Osorio, J., Khan, L., Brandt, P., D'Orazio, V.: ConfliBERT: a pre-trained language model for political conflict and violence. In: Proc. of NAACL-HLT, pp. 5469–5482. Seattle, United States (2022)

23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Larionov, D., Shelmanov, A., Chistova, E., Smirnov, I.: Semantic role labeling with pretrained language models for known and unknown predicates. In: Proc. RANLP, pp. 619–628. INCOMA Ltd., Varna, Bulgaria, September 2019
25. Li, F., Peng, W., Chen, Y., Wang, Q., Pan, L., Lyu, Y., Zhu, Y.: Event extraction as multi-turn question answering. In: Findings of the ACL: EMNLP 2020, pp. 829–838 (2020)
26. Liu, Y., et al.: RoBERTa: a robustly optimized BERT pretraining approach (2019), arXiv: 1907.11692 [cs.CL]
27. Lu, J., Roy, J.: Universal Petrarch: Language-agnostic political event coding using universal dependencies (2017)
28. Maurício, J., Domingues, I., Bernardino, J.: Comparing vision transformers and convolutional neural networks for image classification: A literature review. Appl. Sci. **13**(9), 5521 (2023)
29. Mausam, Schmitz, M., Soderland, S., Bart, R., Etzioni, O.: Open language learning for information extraction. In: Proc. of EMNLP-Comput. NLP, pp. 523–534. Jeju Island, Korea (2012)
30. Munia, M.S., Hosseini, M., Nourani, M., Harvey, J.: Interictal epileptiform discharge detection using time-frequency features and transfer learning. In: Annu. Int. Conf. of the IEEE Eng. in Med. & Bio. Society (EMBC) (2024)
31. Munia, M.S., Hosseini, S.M., Nourani, M., Harvey, J., Dave, H.: Imbalanced EEG analysis using one-shot learning with siamese neural network. In: 2021 IEEE Int. Conf. on Healthcare Informatics (ICHI) (Aug 2021)
32. Munia, M.S., Nourani, M., Harvey, J., Dave, H.: Interictal epileptiform discharge detection using multi-head deep convolutional neural network. In: Annu. Int. Conf. of the IEEE Eng. in Med. & Bio/ Society (EMBC) (2023)
33. Munia, M.S., Nourani, M., Houari, S.: Biosignal oversampling using wasserstein generative adversarial network. In: 2020 IEEE Int. Conf. on Healthcare Informatics (ICHI), pp. 1–7 (2020)
34. Norris, C., Schrodt, P., Beieler, J.: PETRARCH2: another event coding program. J. Open Source Softw. **2**(9), 133 (2017)
35. O'Connor, B., Stewart, B.M., Smith, N.A.: Learning to extract international relations from political context. Proc. 51st Annu. Meeting of the ACL **1**, 1094–1104 (2013)
36. Osorio, J., Reyes, A.: Supervised event coding from text written in Spanish: Introducing eventus id. Soc. Sci. Comput. Rev. **35**(3), 406–416 (2017)
37. Osorio, J., Reyes, A., Beltrán, A., Ahmadzai, A.: Supervised event coding from text written in arabic: Introducing hadath. In: Proc. Workshop on Automated Extraction of Socio-political Events from News 2020, pp. 49–56 (2020)
38. Paszke, A., et al.: PyTorch: An imperative style, high-performance deep learning library. In: NeurIPS, pp. 8024–8035 (2019)
39. Pedregosa, F., et al.: Scikit-learn: Machine learning in python. J. machine learning research **12**(Oct), 2825–2830 (2011)
40. Radford, A., Narasimhan, K.: Improving language understanding by generative pre-training (2018)
41. Radford, B.J.: Multitask models for supervised protests detection in texts. arXiv preprint arXiv:2005.02954 (2020)
42. Rajapakse, T.: Simple transformers (2019). https://simpletransformers.ai/[accessed 2022-08-25]

43. Schrodt, P.A.: Precedents, progress, and prospects in political event data. Int. Interactions **38**(4), 546–569 (2012)
44. Shi, P., Lin, J.: Simple bert models for relation extraction and semantic role labeling (2019). arXiv: 1904.05255 [cs.CL]
45. Skorupa Parolin, E., Hosseini, M., Hu, Y., Khan, L., Brandt, P.T., Osorio, J., D'Orazio, V.: Multi-CoPED: a multilingual multi-task approach for coding political event data on conflict and mediation domain. In: Proc. AAAI/ACM Conf. on AI, Ethics, and Society, pp. 700–711. New York (2022)
46. Stanovsky, G., Michael, J., Zettlemoyer, L., Dagan, I.: Supervised open information extraction. In: Proc. of NAACL-HLT, pp. 885–895 (2018)
47. Touvron, H., et al.: Llama: Open and efficient foundation language models (2023). arXiv: 2302.13971 [cs.CL]
48. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
49. Wang, Y., Qu, W., Ye, X.: Selecting between bert and gpt for text classification in political science research (2024). arXiv: 2411.05050 [cs.CL]
50. Wolf, T., et al.: HuggingFace's transformers: State-of-the-art natural language processing. ArXiv **abs/1910.03771** (2019). arXiv:1910.03771 [cs.CL]
51. Yu, H., Yang, Z., Pelrine, K., Godbout, J.F., Rabbany, R.: Open, closed, or small language models for text classification? (2023). arXiv: 2308.10092 [cs.CL]