ORIGINAL ARTICLE

Knowledge enhanced prompting for few-shot named entity recognition

Zhaoli Liu[1] · Tao Qin[2] · Bohao Liu[2] · Qindong Sun[3,4] · Shancang Li[5]

Received: 15 December 2024 / Accepted: 5 November 2025

Abstract
Named entity recognition (NER) refers to recognize entity spans from text and categorize them into predefined entity types, which is significant for tasks such as knowledge graph construction. However, most existing entity recognition methods perform poorly in professional domains, such as cybersecurity, since they rely heavily on large amounts of high-quality labeled data for model training. In this paper, we introduce prompt-based learning and reformulate the NER task into a generation problem, and propose a knowledge enhanced prompting framework for few-shot NER, which can identify the named entities based on few labeled data. Firstly, we present a knowledge-enhanced template generation method, which integrates the entity domain knowledge and uses BART to generate templates automatically, which helps to overcome the difficulties caused by manual template engineering. Secondly, we construct prompts for each entity type and feed them into the masked language model (MLM) for entity prediction, which do not need to enumerate all the candidate spans, thus reducing the computational complexity. Thirdly, an interaction detection module is designed for entity prediction to constrain the identified entity spans in the input text, which can reduce the uncertainty and unstructured problem for NER. Finally, we perform comparison experiments with existing competitive models on three public datasets and a cybersecurity dataset constructed for professional domain, and the experimental results demonstrate that our method outperforms the existing models on both rich-resource and few-shot settings.

Keywords Few-shot NER · Prompt-based learning · Knowledge enhanced template generation · Interaction detection module

1 Introduction

Named entity recognition (NER) is a fundamental task in natural language processing (NLP), which aims to identify entity spans from text and classify them into pre-defined entity types, such as location, person, organization, etc. It is widely viewed as an important component for tasks such as relation extraction, text summarization, question answering and information retrieval [1]. The current mature solution for handling NER problem is using pre-trained language models (PLMs) combined with task-specific fine-tuning methods[2, 3]. However, the downstream tasks usually introduce a large number of parameters, which need to be trained based on large amounts of high-quality annotated

✉ Tao Qin
qin.tao@mail.xjtu.edu.cn

Zhaoli Liu
zhaoliliu@xaut.edu.cn

Bohao Liu
bhaoliu@163.com

Qindong Sun
sqd@xaut.edu.cn

Shancang Li
shancang.li@ieee.org

1 School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China

2 School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

3 Shaanxi Key Laboratory of Network Computing and Security, Xi'an University of Technology, Xi'an 710048, China

4 Sichuan Digital Economy Industry Development Research Institute, Chengdu 610036, China

5 School of Computer Science and Informatics, Cardiff University, Cardiff G10157, UK

Published online: 02 February 2026

data. Unfortunately, obtaining high-quality NER annotations in professional domains, such as cybersecurity, can be quite expensive [4].

Few-shot learning is one of the technologies that can solve the challenges posed by the limitation of annotated data. The few-shot NER aims to identify named entities based on a few labeled data, such as prototype-based methods [5, 6]. The main idea of these methods is to learn a prototype representation for each entity type, and then compute the distances between the samples and the prototype representations, and finally classify the samples to the nearest prototype. However, these methods usually learn a noisy prototype to represent the “O” type (non-entity type), which affects their performance in few-shot NER [7].

Prompting, also known as prompt-based learning, has lately achieved great success in few-shot classification tasks by reformulating the downstream task as a masked language modeling problem, the model input is converted into a prompt-based input with some unfilled masked tokens through a template, and then PLMs are used to predict these masked tokens [8]. Prompting holds the same objective form as pre-training, thus no new parameters are introduced, it can quickly adapt to different downstream tasks. But for NER tasks, the entity spans and entity types need to be identified simultaneously, and both of them are not fixed in length; moreover, there may be no entity or multiple entities in the input text, which makes it difficult to apply prompting to NER tasks.

Recently, some researchers try to develop prompt-based NER methods [9]. Cui et al. [10] proposed a template-based method for few-shot NER (abbreviated as TemplateNER), they manually design a series of templates, and enumerate all the candidate spans in the input text and fill them in the templates, and finally classify each candidate span based on the corresponding template scores. Although this type of methods can obtain remarkable results, thsy still have following limitations: (1) Manually templates design and evaluation, which requires a lot of human effort and domain knowledge [11]. (2) These prompting methods have to enumerate all n-grams tokens to find the possible entities, which suffers from high computational complexity. (3) When evaluating the templates, the objective form of scoring the templates is different from that of pre-training models, which does not conform to prompting paradigm.

To address these challenges, we reformulate the NER task as a generation problem and propose an automatic prompting framework for few-shot NER. Firstly, we propose a knowledge enhanced template generation method, which applies prompting to automatically generate the optimal templates based on the entity knowledge extracted from the labeled data, which helps to overcome the difficulties caused by manual template engineering. Secondly, for the prompt construction, we do not need to enumerate all the possible spans, we just construct prompts for each entity type and then tune the LM in a generative framework, which greatly reduce the computational complexity. Thirdly, we design an interaction detection module for entity prediction to constrain the entity spans in the input texts, which can mitigate the output uncertainty and unstructured problem. Meanwhile, we design special symbols as the separator and terminator to solve the problem posed by one specific sentence contains multiple entities. Finally, we conduct a series of comparison experiments on three public NER datasets, including CoNLL03, MIT Movie and MIT Restaurant, and a cybersecurity dataset constructed as an example for experiments in professional domain. The experimental results demonstrate that our approach outperforms existing competitive models in both rich-resource and few-shot settings. We also performed ablation study to analyze how different factors might affect the performance.

The remainder of this paper is organized as follows. We briefly review the related work in Sect. 2. Section 3 presents the framework of the proposed model. Detailed descriptions of the knowledge-enhanced template generation method, prompt construction, and entity prediction are presented in Sect. 4. The experimental settings and results are presented in Sect. 5, the experimental results are presented in Sect. 6, and the ablation study is discussed in Sect. 7. In Sect. 8, we discuss the advantages and limitations of our model. Finally, Sect. 9 concludes the paper.

2 Related work

With the prevailing of pre-trained language models, especially Transformer-based models, such as GPT [12], BERT [13] and BART [14], the pre-training and fine-tuning paradigm has yielded strong empirical performance on various natural language processing (NLP) tasks. The current dominant methods treat NER as a sequence tagging problem, and use pre-trained language models such as BERT to represent the input text, and then label-specific classifiers or CRF output layers are added to assign entity tags or non-entity tags on each input token [15, 16]. However, these approaches usually rely on large labeled data for model training, which can be available for some general domains such as news, but scarce in most professional domains such as cybersecurity. Unfortunately, obtaining a large number of NER annotations for such professional domains can be quite expensive and often requires a great deal of domain knowledge. Thus, few-shot NER becomes a challenging but practical research problem and receives a lot of attention.

For few-shot NER, one line of research is prototype-based methods, which involve meta-learning and have become popular in the NER area. Snell et al. [5] proposed prototypical networks for few-shot learning, which learn a metric space during training, and then classification can be performed by computing distances between the sample and prototype representations of each class. Fritzler et al. [6] and Hou et al. [17] applied prototypical networks to few-shot NER and utilized the nearest-neighbor criterion to assign the entity type. However, these approaches label all non-entity spans as the same class Outside (O), thus leading to learn a noisy prototype representation for “O” class. Therefore, instead of learning a prototype for each entity class, Yang et al. [18] represented each token in the labeled examples of the support set by its contextual representation in the sentence. Das et al. [19] introduced contrastive learning during training, in-batch examples are considered as positive examples if they have the same label, and others are taken as negative, which can further optimize the learning of the representation space. However, these methods based on metric learning might be less effective when encountering a large domain gap, since they just directly use the learned metric without any further adaptation to the target domain.

Recently, prompt-based methods have achieved significant performance in few-shot learning scenarios by bridging the gap between pre-training and fine-tuning for downstream tasks, in which the model input is converted into a prompt-based input with some unfilled masked tokens through a template, and then PLMs are used to predict these masked tokens [20]. The effectiveness of prompt learning has been verified in various NLP tasks, including text classification [21, 22], relation extraction [23], translation [24], and sentiment classification [25]. However, there only have a few studies on NER, especially in the few-shot setting. For the NER task, there are two elements need to be predicted: entity spans and entity types, both of them are variable in length, and the input text may contain no entity or multiple entities, which makes applying prompting methods to NER more challenging. Cui et al. [10] employed manually designed templates for prompting in few-shot NER, but this approach requires manual template engineering and involves enumerating all possible spans in the input text, which is computationally expensive. To overcome this limitation, Ma et al. [26] proposed a template-free model that alleviates the cost of enumerating spans, but there is still an inconsistency between their training objective and PLMs, which may result in model confusion. Ye et al. [27] proposed a prompt-based framework that decomposes the NER task into two sub-tasks, entity locating and entity typing. By integrating sequence labeling with prompting, their method improves efficiency compared to previous approaches. Shen et al. [28] further optimized prompt construction by unifying entity locating and entity typing into a single-round prompt learning process. Huang et al. [29] proposed a template based on label words and an example-based template, inserting special tokens in prompts as boundary markers for entity prediction.

However, the existing prompt-based methods have several limitations, such as the need of manual template engineering, the issues of time consumption and inconsistency with pre-trained training objectives. Enlightened by the related works and focus on addressing these challenges, we introduce the automatic template generation and prompt construction optimization to NER task, which can offer a more accurate and efficient solution for low-resource scenarios.

3 Overview of the proposed model

3.1 Task definition

Given an input text X of l tokens, $C = [c_0, ..., c_{k-1}]$ denote the entity type set (such as “LOC”, “ORG”, etc.), k is the number of entity types, the NER model intends to extract all the span pairs $\{(e_i, c_i)\}$, where e_i is the entity span and c_j is the entity type. In few-shot settings, model are often evaluated on multiple low-resource domains, each domain only contains a few labeled instances called support set S, which usually includes n examples (n-shot) for each of k entity types. Our goal is to train an accurate NER model with prompting learning under low-resource scenarios, which can solve the problem caused by the sparse labeled data in professional domains.

3.2 Framework of the proposed model

The knowledge enhanced prompting framework for few-shot NER is shown in Fig. 1, which is mainly composed of automatic template generation module, prompt construction module and entity prediction module. In order to solve the problem that the entity spans are not fixed in length, we employ the Encoder-Decoder model BART as the base model. The developed model mainly contain three components as follows:

- Automatic template generation. We use PLM to automatically generate the optimal templates based on the ground-truth entities in the labeled data, thus we do not need the manual template engineering to design and select the suitable templates.
- Prompt construction. We construct prompts for each entity type, and then send them to PLM for entity prediction. In this way, we do not need to enumerate all possible spans, which can greatly reduce the computational complexity.

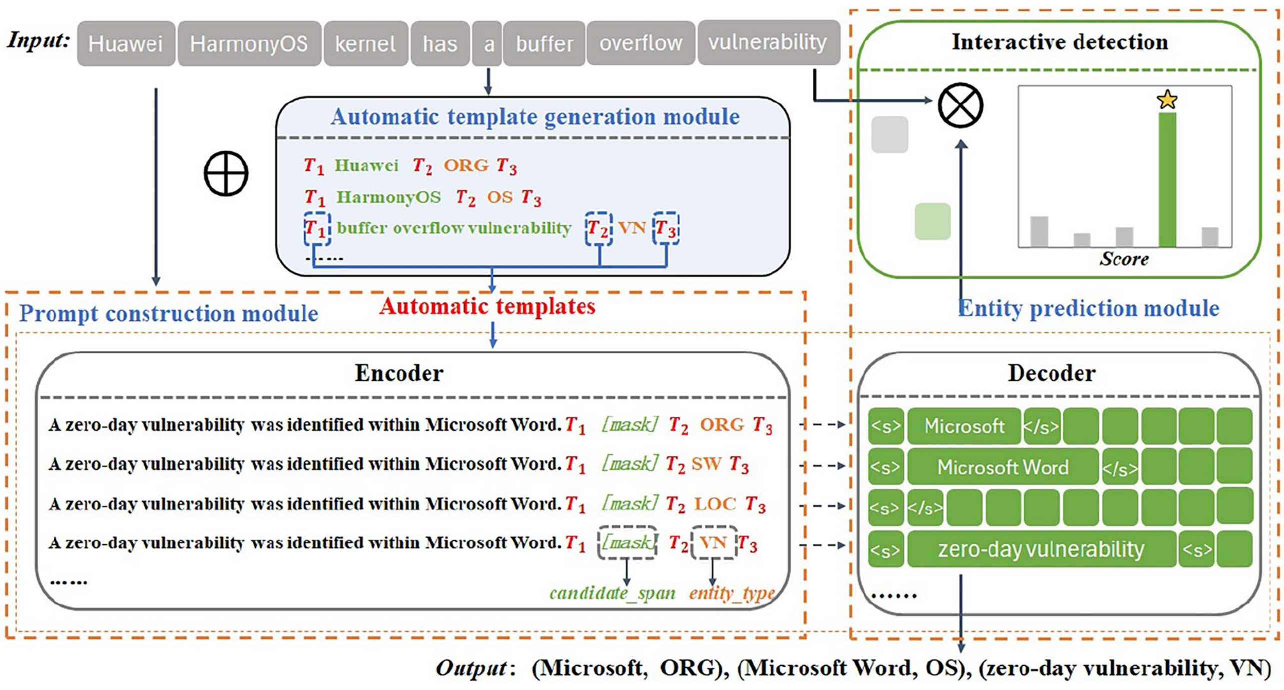
Input: Huawei HarmonyOS kernel has a buffer overflow vulnerability
Automatic template generation module
T1 Huawei T2 ORG T3
T1 HarmonyOS T2 OS T3
T1 buffer overflow vulnerability T2 VN T3
Automatic templates
Prompt construction module
Encoder
A zero-day vulnerability was identified within Microsoft Word. T1 [mask] T2 ORG T3
A zero-day vulnerability was identified within Microsoft Word. T1 [mask] T2 SW T3
A zero-day vulnerability was identified within Microsoft Word. T1 [mask] T2 LOC T3
A zero-day vulnerability was identified within Microsoft Word. T1 [mask] T2 VN T3
candidate_span entity_type
Interactive detection
Score
Entity prediction module
Decoder
<s> Microsoft </s>
<s> Microsoft Word </s>
<s> </s>
<s> zero-day vulnerability <s>
Output: (Microsoft, ORG), (Microsoft Word, OS), (zero-day vulnerability, VN)

Fig. 1 Framework of the proposed model. Totally, there are three modules, the upper-left part is the automatic template generation module and the lower-left part is the prompt construction module, the right part describes the entity prediction module

- Entity prediction. We design an interaction detection module for entity prediction, which employs the hidden vector of the decoder and the encoding vector of the input text for score calculation, and then chooses the token with the highest score as the predicted entity. In this way, we can make sure that the predicted entity spans are extracted from the input text, in turn, solve the uncertainty and unstructured problem in NER task.

The proposed model does not introduce any additional parameters, and its training objective is completely consistent with that of the pre-trained model, which can better utilize the knowledge from pre-trained language model.

4 Methodology

4.1 Knowledge enhanced template generation

For named entity recognition task, there are three elements to build a prompt: input text X, entity type $[entity\_type]$ and the entity span to be extracted $[candidate\_span]$. For the input text X, the constructed prompt can be represented as:

$$X_{prompt} = [CLS]\ X\ T_1\ [candidate\_span]\ T_2\ [entity\_type]\ T_3\ [SEP] \tag{1}$$

Table 1 Examples of manually designed templates

Templates	T_1	T_2	T_3
Huawei is organization	None	is	None
Huawei belongs to organization type	None	belongs to	type
Huawei's entity type is organization	None	's entity type is	None
Entity Huawei is an organization	Entity	is an	None
Huawei should be tagged as organization	None	should be tagged as	None

where $[CLS]$ and $[SEP]$ are two special tokens indicating the start and end of the input text, and T_i represents the prompt templates, which can be manually designed. For instance, for a cybersecurity text "Huawei HarmonyOS kernel has a buffer overflow vulnerability.", we can design templates as shown in Table 1:

For the manually designed templates in Table 1, $[candidate\_span]$ and $[entity\_type]$ are "Huawei" and "organization" respectively. Manually designed templates can be flexible, but experiments indicate that different templates have a great influence on the final entity recognition results, and the manual templates are not likely to be optimal. Therefore, we propose an automatic template generation method, which can integrate the entity knowledge from the labeled data into the prompting, thus making the templates more closely related to NER task.

For the traditional prompt-based NER methods, the templates are pre-designed, and the masked positions are to be filled or predicted by the LMs. In contrast, we

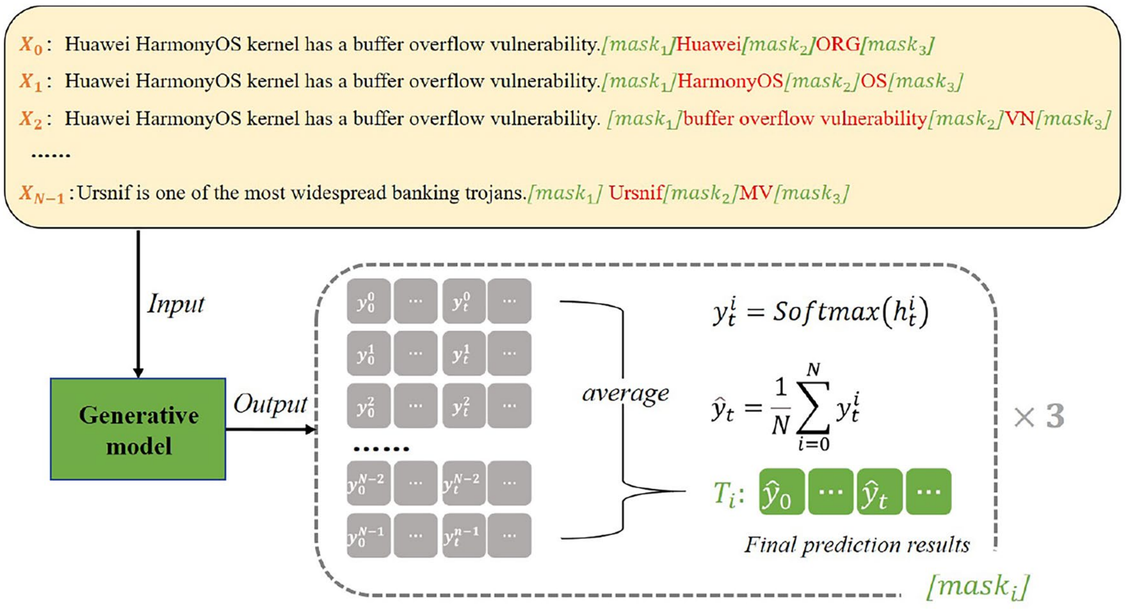
X_0: Huawei HarmonyOS kernel has a buffer overflow vulnerability.[$mask_1$]Huawei[$mask_2$]ORG[$mask_3$]
X_1: Huawei HarmonyOS kernel has a buffer overflow vulnerability.[$mask_1$]HarmonyOS[$mask_2$]OS[$mask_3$]
X_2: Huawei HarmonyOS kernel has a buffer overflow vulnerability. [$mask_1$]buffer overflow vulnerability[$mask_2$]VN[$mask_3$]
......
X_{N-1}: Ursnif is one of the most widespread banking trojans.[$mask_1$] Ursnif[$mask_2$]MV[$mask_3$]
Input
Generative model
Output
average
$y_t^i = Softmax(h_t^i)$
$\hat{y}_t = \frac{1}{N}\sum_{i=0}^{N} y_t^i$
T_i: $\hat{y}_0$ ... $\hat{y}_t$ ...
Final prediction results
× 3
[$mask_i$]

Fig. 2 Knowledge enhanced template generation

rebuild the prompt for the automatic template generation as shown in Fig. 2, the $[entity\_type]$ and its corresponding $[candidate\_span]$ are known based on the entity knowledge from the labeled data, and the template positions are masked out and sent to BART for prediction.

For the input text X, the prompt we constructed as shown in Eq. (2):

$$X_{prompt} = [CLS]\ X\ [mask1]\ candidate\_span\ [mask2]\ entity\_type\ [mask3]\ [SEP] \tag{2}$$

where $candidate\_span$ and $entity\_type$ are the entity spans and entity types from the labeled data, we construct the prompt-based input X_{prompt} with some unfilled masked template tokens, and then send to BART model to predict the masked template tokens [*mask*1], [*mask*2] and [*mask*3] respectively. Here we apply Beam Search to generate the prediction, taking template position [*mask*1] as an example, assuming the output vector from the last hidden layer of the decoder is h_t, the model prediction is a multi-class classification task:

$$y_t = Softmax(h_t) \tag{3}$$

where $y_t \in R^v$ indicates the probability that $token_t$ belongs to each word/phrase, and v is the vocabulary size of the LMs. We construct X_{prompt} sequence $(X_1, X_2, \ldots, X_N)$, and send to the model for prediction, then we can obtain the prediction probability y_t^i for each input X_i, and finally the average value of N predictions is taken as the predicted probability $\hat{y}_t$ of $token_t$ as follows:

Table 2 Examples of automatically constructed templates

Templates	T_1	T_2	T_3
(Huawei) organization.	(	)	.
This is Huawei's organization.	This is	's	.
The Huawei in organization.	The	in	.
. Huawei in organization.	.	in	.
The Huawei's organization.	The	's	.

$$\hat{y}_t = \frac{1}{N}\sum_{i=0}^{N} y_t^i \tag{4}$$

We choose the top k tokens with the highest probability as candidate prediction results, then each token is regarded as the current decoding result and sent to the model for the subsequent prediction at $t+1$. Finally we obtain the prediction sequence $Y = (Y_1, Y_2, \ldots, Y_i, \ldots)$, where $Y_i = (\hat{y}_0, \ldots, \hat{y}_t, \ldots)$, the sequence with the highest probability is taken as the automatic template T_1. For template tokens [*mask*2] and [*mask*3], we use the same way to obtain T_2 and T_3.

The top 5 templates with the highest probability are shown in Table 2. Here we still use the example illustrated in Table 1, where $[candidate\_span]$ and $[entity\_type]$ are "Huawei" and "organization" respectively.

4.2 Prompt construction and learning

By using the automatic generated templates, we can construct prompts for each entity type and feed them into the MLM model. In this paper, we select BART as the pre-trained generative language models. For prompt construction, we fill entity type into the $[entity\_type]$ position, here the entity type is used as one part of the prompt, so we just need to mask out the entity spans. For instance, given the input text *X*, we can use the automatic template to construct prompt for ORG (organization) entity type as following:

$$X_{prompt} = [CLS]\ X\ (\ [mask]\ )\ ORG\ [SEP] \tag{5}$$

We construct prompts for each entity type, and then guide the model to predict specific types of entities. Given a sentence *X* of length *l*, we fill a fixed number *k* of prompts and *X* into the automatic template to construct X_{prompt}, and *k* is the number of entity types. The model then fills the masked slots of all prompts and decodes the named entities in the sentence. Usually the input text contains only a few named entities, or even no named entities. If we construct X_{prompt} for each entity type during the training phase, the prompts which do not contain any entity would account for a significant proportion, we treat them as negative samples. As the number of entity types increases, the proportion of negative samples would become larger, which may cause a serious class imbalance problem, and the model would be more inclined to predict that the text does not contain any entity. To address this issue, we set a hyperparameter—negative sampling rate α, $0 < \alpha < 1$, the number of negative samples that are fed to the model during the training phase is defined as the total number of negative samples multiplied by α.

We feed the prompt X_{prompt} to the encoder of the BART, and then we obtain the hidden representations as follows:

$$H_{en} = Encoder(X_{prompt}) \tag{6}$$

where $H_{en} \in R^{l'*\ d}$, l' is the length of the input sequence, and *d* is the dimension of the hidden layer vector.

4.3 Interaction detection module for entity prediction

For the prediction of token y_t at time *t*, the encoding vector H_{en} of the encoder and the previous predictions $y_1, y_2, \ldots, y_{t-1}$ are used as the input of the decoder, then BART model obtains the output vector h_t from the last hidden layer, and finally the Softmax is utilized to obtain the final prediction results y_t:

$$h_t = Decoder(H_{en}; y_1, y_2, \ldots, y_{t-1}) \tag{7}$$

$$y_t = Softmax(h_t) \tag{8}$$

where $y_t \in R^v$, *v* is the size of the vocabulary. However, for NER task, $[candidate\_span]$ can only extract from the input text *X*. To constrain the entity spans in the input text, we design an interaction detection module, the output vector h_t from the last hidden layer is calculated with the input text *X* in the interaction detection module to obtain the final prediction results y_t. Specifically, after obtaining the encoding vector H_{en} for X_{prompt}, the first *l* tokens are selected as the input of the MLP layer.

$$H'_{en} = MLP(H_{en}[:l]) \tag{9}$$

where $H'_{en} \in R^{l*d}$. In addition, we introduce a separator symbol x_{gap}, which represents the entity gap when there are multiple entities of the same entity type in the input text, and its token encoding h_g can be calculated as follows:

$$H_g = TokenEmbed(x_{gap}) \tag{10}$$

where $h_g \in R^d$. We concatenate H'_{en}, h_g and the hidden layer vector h_s of the terminator $x_{</s>}$ to get the final candidate sequence encoding vector $\hat{H}$.

$$\hat{H} = [H'_{en}; h_g; h_s] \tag{11}$$

where $\hat{H} \in R^{(l+2)*d}$. We calculate the dot product of h_t and $\hat{H}$, and apply a Softmax layer to obtain the subscript i_t of token y_t in the input sequence, and the conditional probability of i_t can be calculated as follows:

$$p(i_t|y_{0:t-1}) = Softmax(h_t \otimes \hat{H}) \tag{12}$$

where $0 \le i < l+2$, and y_t can be obtained after encoding:

$$y_t = \begin{cases} x_{i_t}, & if \quad i_t < l \\ x_{gap}, & if \quad i_t = l \\ x_{</s>}, & if \quad i_t = -1 \end{cases} \tag{13}$$

We define the special token "<s>" as the start token, which is concatenated with the original input sequence as the input of decoder, then we have $y_0 = x_{<s>}$. Teacher forcing is applied to the model training, and the loss function is defined as the negative log-likelihood function, as shown in the following formula:

$$L = \sum_{t=0}^{m} \log p(i_t|y_{0:t-1}, X_{prompt}) \tag{14}$$

where *m* is the length of the output. In the prediction phase, the token predicted at time point $t-1$ is used as the input of decoder at time point *t*, and the encoding ends when encountering the terminator "</s>".

For the case where there are multiple entities of the same type, we design special symbol "<gap>" as the separator of decoding. Figure 3 shows an example of the decoding results for the input text "Tencent discovered a high-risk vulnerability in Huawei HarmonyOS". For the OS (Operating System) entity type, the output of decoder should be "<s>HarmonyOS< /s>c. If the input text does not contain the entity type, i.e. for LOC (location) entity type, then the output should be "<s>< /s>". For ORG (organization) entity type, there are two entity spans, the output should be "<s>Tencent<gap>Huawei< /s>".

We construct prompts for each entity type and feed into BART, and then send to interaction detection module for entity prediction. For each prompt X_{prompt}, we can only get the entity spans corresponding to the given entity type. Therefore, for the input text X, the NER results should be the sum of the decoding results for all the X_{prompt} sequences. The entity prediction process for the input text X is described in Algorithm 1:

Algorithm 1 Entity Prediction

```
Input: The output sequence of Decoder $Y = [Y_0, Y_1, \dots, Y_{k-1}]$, the entity types $C = [c_0, c_1, \dots, c_{k-1}]$
Output: Entity spans $E = (e_0, c_0), (e_1, c_1), \dots, (e_n, c_n)$
1:  Initialization $E = \{\}$, $e = []$
2:  for $i$ in $(0, k)$ do
3:      $(y_0, y_1, \dots, y_{m-1}) = Y_i = Y[i]$; $c_i = C[i]$
4:      Initialization $t = 0$
5:      while $t < m$ do
6:          $y_t = Y_i[t]$
7:          if $y_t == x_{gap}$ then
8:              $E$.add(($e$,$c_i$))
9:              $e = []$
10:         else
11:             $e$.append($y_t$)
12:         end if
13:     end while
14:     if len($e$) > 0 then
15:         $E$.add(($e$,$c_i$))
16:     end if
17: end for
18: return $E$
```

5Experiment settings

5.1 Datasets

In order to validate the effectiveness of the proposed model in general domains, we use three famous public NER datasets for evaluation. We choose the CoNLL03 [30] as the resource-rich dataset, MIT Movie Review [31] and MIT Restaurant Review [31] as the few-shot datasets.

Meanwhile, to evaluate the proposed model in professional domains such as cybersecurity, we collected expert analysis reports from open-source websites and constructed a cybersecurity dataset (abbreviated as CSNER). Due to the high specialization of cybersecurity domain, for example, it is difficult to identify the technical term "RedCurl" as an organization entity, a software name, or a malware without sufficient contextual information and domain knowledge. Therefore, we collected the cybersecurity terms, such as vulnerabilities, softwares, hardwares and so on, to construct an entity dictionary to introduce relevant domain knowledge, and then developed a dictionary-based approach and manual annotation to complete the annotation task. We totally identified 13 entity types, namely: Software (SW), Hardware (HW), Operating System (OS), Version (VER), Location (LOC), Organization (ORG), Vulnerability ID

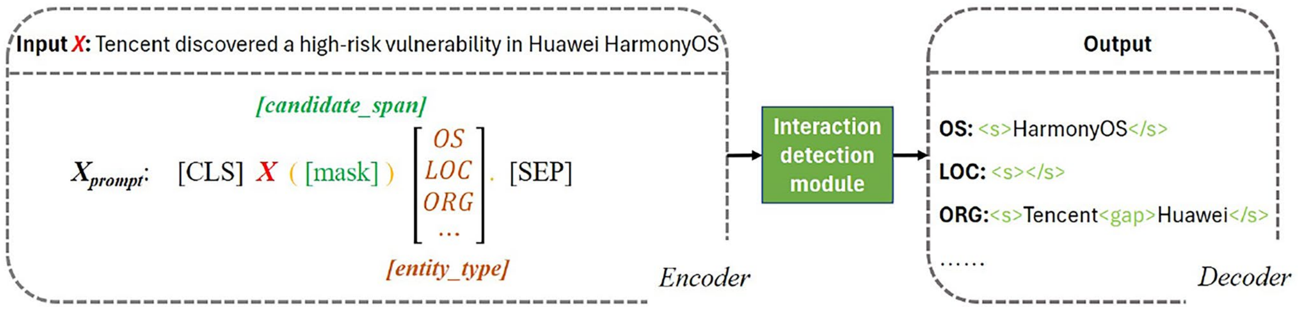
Input X: Tencent discovered a high-risk vulnerability in Huawei HarmonyOS
[candidate_span]
Xprompt: [CLS] X ([mask]) [OS LOC ORG …]. [SEP]
[entity_type]
Encoder
Interaction detection module
Output
OS: <s>HarmonyOS</s>
LOC: <s></s>
ORG:<s>Tencent<gap>Huawei</s>
……
Decoder

Fig. 3 The decoding of interaction detection module. For the cases where there are multiple entities of the same type, such as <s>Tencent<gap>Huawei< /s>, "<gap>" is defined as the separator of decoding, "<s>" and "</s>" are the start-of-sentence and end-of-sentence tokens

(VI), Vulnerability Type (VN), Attacker (AR), Attack Type (AT), Malware (MW) , Consequence (CSQ), Malware Type (MWT). The CSNER dataset is publicly available via https://github.com/fantomek/security_ner. Details of the datasets are shown in Table 3.

To evaluate the proposed model on few-shot settings, we conduct experiments with $n = (10, 20, 50, 100, 500)$, which means there are only n samples of each entity type available for training. For NER task, each sampled sentence might include multiple entities with different types, randomly selecting sentences will cause a serious imbalance in the training dataset. In our experiments, we develop a few-shot sampling strategy to ensure that we exactly obtained n samples for each entity type under the n-shot experimental settings. We first count the number for each entity type in the sentences and then sort the sentences in an ascending order. Then we select training samples for each entity type in sequence. For entity type c_i, if the selected sentence s_j contains entity type c_i, then we add it into the few-shot NER dataset and count all other entities in it, until the number of samples for entity type c_i reaches n. We use the same way to select training samples for entity type c_{i+1}. The details of the sampling algorithm for few-shot data construction are given in Algorithm 2.

Algorithm 2 Few-shot sampling

```
Input: # of shot $n$, # of entity types $k$, the sequences in dataset $D$, entity types
    sorted by size from smallest to largest $Order$
Output: $n$-shot NER dataset: $S = [s_0, s_1, \ldots, s_N]$
 1: Initialization $n$-shot NER dataset $S$={}, the dictionary that counts the number
    for each entity type $Count = \{c_0 : 0, c_1 : 0, \ldots c_k : 0\}$, $i = 0$
 2: while $i < k$ do
 3:     $c_i = Order[i]$
 4:     while $Count[c_i] < n$ do
 5:         Randomly select sentence $d_j$ from the dataset $D$
 6:         if $d_j$ contains entity type $c_i$ then
 7:             $S$.add($d_j$)) //Add $d_j$ to $n$-shot NER dataset
 8:             for $e, c$ in $d_j$ do
 9:                 $Count[c] = Count[c] + 1$
10:             end for
11:         end if
12:     end while
13:     $i = i + 1$
14: end while
15: return $S$
```

Table 3 Details of the datasets

Datasets	CoNLL03	MIT restaurant	MIT movie	CSNER
#Train	14,987	7660	7816	6042
#Test	3466	1521	1953	1067
#Entity types	4	8	12	13

5.2 Baselines and experimental settings

In our experiments, we mainly compare our method with four representative baselines, involving both sequence labeling and prompt-based approaches.

- 1. *Sequence Labeling BERT* [16]: A representative sequence labeling model, which uses BERT as Encoder to learn the sequence representation, and then sent to the BiLSTM-CRF model for prediction.
- 2. *TemplateNER* [10]: A template-based prompting method, which uses the manually designed templates, and enumerates all possible spans in the input sentence to construct prompts, finally classifying them based on BART scores on the templates.
- 3. *UnifiedNER* [32]: A unified generative framework which uses BART as the pre-trained model, and adopts a generative solution to solve the flat NER and nested NER tasks.
- 4. *PromptNER* [28]: A prompt-based method which unifies entity locating and entity typing into prompting, and designs a dynamic template filling mechanism to assign labels for the position and type slots by bipartite graph matching.

Table 4 Details of the hyperparameter settings

Hyperparameters	Definition	Value
BART embedding	The dimension of character feature vector	768
MLP hidden size	The dimension of MLP layer vector	128
Max seq length	The maximum length of training sentence	128
Batch size	The number of training samples per batch	2/4/8/16/32
Learning rate	Initial learning rate	1e–5
Dropout rate	The rate of randomly dropping units	0.9
Num train epochs	The number of training times for all the samples	40
Warmup proportion	Warmup learning rate	0.01
Max_length	The maximum length of model output	10
Max_length_a	The maximum length ratio of model output	0.6
Negative ratio	Negative sampling rate	0.5

We also investigate the impact of different templates by choosing the manual template "<candidate_span> is a <entity_type> entity." and the automatic template "(<candidate_span>) <entity_type>.".

The proposed model in this paper introduces no new parameters except for the MLP layer, the model parameters are consistent with the pre-trained model BART-large, which uses a deep 12-layer encoder and decoder. The hyperparameter settings are shown in Table 4. The batch sizes listed are intended to indicate the range of batch sizes used during experiments? which is set to 32 in the rich-resource scenario and adjusted according to the available training samples in few-shot scenarios.

6 Experimental results

6.1 Results of rich-resource settings

We first compare the proposed model with the baselines under the standard NER setting on CoNLL03. The experimental results are shown in Table 5.

As can be seen from Table 5, the sequence labeling BERT gives a strong baseline on rich-resource settings, F1 score reaches 92.93%. Even though TemplateNER method is designed for few-shot named entity recognition, it also performs better than the sequence labeling BERT on rich-resource settings, and since it uses sliding windows to enumerate all the candidate entity spans, which achieves a high recall rate, reaching 94.34%, but its precision rate only reaches 91.89%, lower than the sequence labeling BERT. UnifiedNER method has a relatively balanced performance in precision and recall rate, with the F1 score slightly higher than the previous two models. PromptNER method optimizes the prompt construction, thereby improving its efficiency, but its performance remains basically equivalent to that of TemplateNER under rich-resource conditions. Our model shows the best performance on rich-resource settings, which demonstrates the effectiveness of our model in NER, and its performance is further improved when using automatically generated template, indicating that the automatically generated template is more effective than the manually designed template.

Table 5 Rich-resource NER results on CoNLL03 (batch size is set to 32)

Methods	P (%)	R (%)	F1 (%)
Sequence labeling BERT	93.12	92.74	92.93
TemplateNER	91.89	94.34	93.10
UnifiedNER	92.61	93.87	93.24
PromptNER	92.96	93.18	93.08
Our method (manual template)	93.16	93.62	93.39
Our method (automatic template)	93.98	94.86	94.42

Table 6 Few-shot NER results on MIT restaurant

Methods	F1 score (%)					
	10	20	50	100	200	500
Sequence labeling BERT	21.80	39.40	52.70	53.50	57.40	61.30
TemplateNER	46.00	57.10	58.70	60.10	62.80	65.00
UnifiedNER	38.39	53.33	60.53	66.40	73.03	73.53
PromptNER	56.10	62.60	69.30	71.30	74.40	77.40
Our method (manual template)	53.24	60.78	67.95	71.36	75.88	76.02
Our method (automatic template)	59.67	65.39	71.40	74.70	77.52	78.84

Batch size is set to 2 for the 10-shot and 20-shot settings, 4 for the 50-shot and 100-shot settings, 8 for 200-shot and 16 for 500-shot setting

Table 7 Few-shot NER results on MIT movie

Methods	F1 score (%)					
	10	20	50	100	200	500
Sequence labeling BERT	25.20	42.20	49.64	50.70	59.30	74.40
TemplateNER	37.30	48.50	52.20	56.30	62.00	74.90
UnifiedNER	55.29	62.68	73.95	77.40	82.06	85.18
PromptNER	55.60	68.20	76.50	80.40	82.90	84.50
Our method (manual template)	63.25	70.38	72.88	76.74	79.96	82.43
Our method (automatic template)	72.25	76.50	79.09	81.70	84.05	86.07

Batch size is set to 2 for the 10-shot and 20-shot settings, 4 for the 50-shot and 100-shot settings, 8 for 200-shot and 16 for 500-shot setting

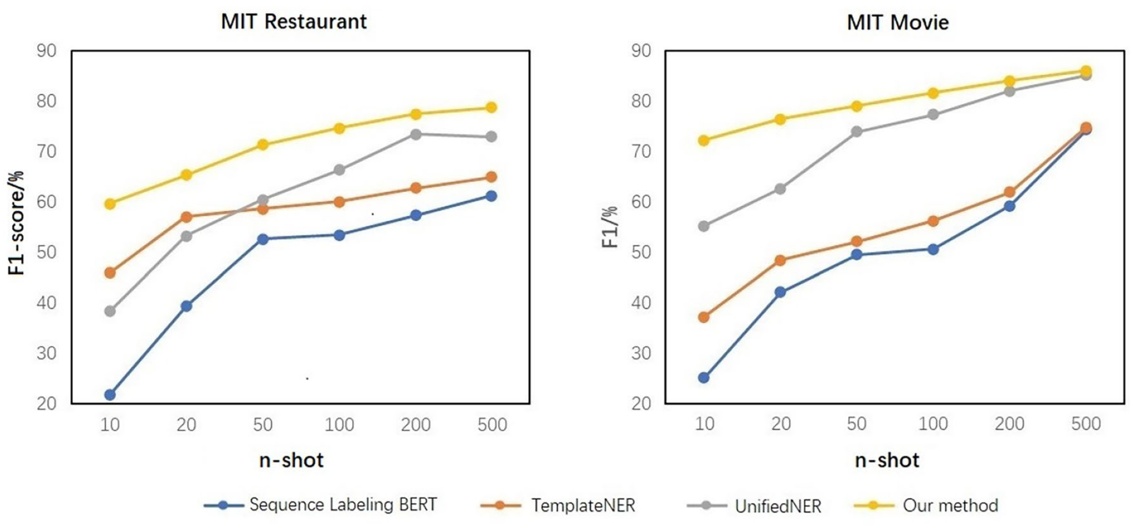

Fig. 4 Performance of different number of training samples

6.2 Results of few-shot settings

To evaluate the model performance under few-shot learning scenario in general domain, we first build n-shot NER datasets from MIT Restaurant and MIT Movie respectively according to Algorithm 2. For each entity type, we sample a fixed number of instances $n = (10, 20, 50, 100, 500)$ from MIT Restaurant and MIT Movie datasets to construct the few-show datasets. If there are only a small number of instances for one entity type, the selection of training samples has a great impact on the evaluation results, so we use Algorithm 2 to construct 5 different few-shot datasets, and the averages of the experimental results on these 5 datasets are employed to evaluate our model. The few-shot experimental results on MIT Restaurant and MIT Movie datasets are given in Tables 6 and 7 respectively.

From Tables 6 and 7, it can be seen that under the few-shot NER scenario, Sequence Labeling BERT performs much worse than other models. For the 10-shot NER, the F1 score of Sequence Labeling BERT on the MIT Restaurant and MIT Movie datasets are as low as about 20%. Due to the introduction of prompt, TemplateNER method achieves a performance improvement of about 25% on MIT Restaurant, and UnifiedNER method improves the F1 score by 76%. For the MIT Restaurant dataset, TemplateNER outperforms UnifiedNER in the 10-shot and 20-shot scenarios, but

Table 8 Few-shot NER results on cybersecurity dataset

Methods	F1 score (%)					
	10	20	50	100	200	500
Sequence labeling BERT	21.20	30.16	40.64	50.70	57.30	60.40
TemplateNER	32.72	41.78	46.27	56.30	62.00	64.90
UnifiedNER	30.16	38.68	43.58	57.40	61.76	64.18
PromptNER	35.27	42.75	50.06	58.38	61.80	65.73
Our method (manual template)	37.30	43.83	49.72	57.34	60.57	65.52
Our method (automatic template)	39.54	46.50	54.09	62.70	64.05	70.86

Batch size is set to 2 for the 10-shot and 20-shot settings, 4 for the 50-shot and 100-shot settings, 8 for 200-shot and 16 for 500-shot setting

as the number of training samples increases, UnifiedNER's performance becomes better than TemplateNER, while for the MIT Movie dataset, UnifiedNER demonstrates superior performance in all scenarios compared to TemplateNER. This might be related to the fact that MIT Movie dataset has more kinds of entity types, the idea of exhausting all the spans in TemplateNER would be time-consuming and less effective when there are too many entity types. PromptNER achieves better performance than TemplateNER and UnifiedNER for the two datasets, which demonstrates that optimizing prompt construction can enhance the performance of few-shot NER. Our model shows a significant improvement over the other models on these two datasets, especially when using automatic template.

Moreover, the F1 score of our model is much higher than TemplateNER and UnifidedNER on MIT Movie dataset, which indicates that the increase of entity types does not affect much on the performance of our model.

We also analyze the performance changing trend with different number of training samples. Figure 4 shows the curve of F1 score for different models as the number of training samples increases. As shown in Fig. 4, our model performs the best under different number of training samples, especially when using the automatic templates. F1 score of each model shows a significant difference when there is less training data, as the training data increases, the F1 scores of these models gradually increase, but our model still outperforms the other models, which also indicates that Sequence Labeling BERT relies on the availability of sufficient training data, so it cannot be well trained under limited labeled data settings, but our model can make better use of the knowledge from pre-trained models, and requires only a small amount of labeled data to complete the training.

6.3 Results of few-shot cybersecurity dataset

In order to validate the proposed model in the professional domains, we construct the cybersecurity dataset CSNER and build n-shot NER datasets according to Algorithm 2 for experiments, where $n = (10, 20, 50, 100, 500)$. The experimental results based on the few-shot datasets are given in Table 8.

Table 9 Experimental results of different templates

No	Templates	P(%)	R(%)	F1(%)
1	<candidate_span> is a <entity_type> entity	57.39	70.07	63.1
2	The entity type of <candidate_span> is <entity_type>	66.21	75.29	70.46
3	The <candidate_span> in <entity_type>	66.73	72.5	69.5
4	This is <candidate_span> 's <entity_type>	64.69	76.54	70.12
5	(<candidate_span>) <entity_type>	67.06	79.31	72.25

As shown in Table 8, the proposed model outperforms the other models on precision, recall and F1 score. For the 10-shot NER, the case of minimal labeled data, our model improves the entity recognition performance by 78% over Sequence Labeling BERT, 21% over TemplateNER and 12% over PromptNER. It can be also seen that all the prompt-based models including TemplateNER, UnifiedNER, PromptNER and our model perform a little worse than that of the MIT Restaurant and MIT Movie datasets, we believe that there are two main reasons: (1) The training data of the pre-trained language models is mainly derived from the general domains, which lack of professional knowledge in cybersecurity domain, so the performance of prompt-based methods is not as good as that achieved on general-domain public datasets. (2) The structure of security entities is more complex, for example, security entities such as vulnerability IDs are composed of words and numbers, and cybersecurity dataset has more entity types than the MIT datasets, which makes the few-shot NER task in cybersecurity domain more challenging.

7 Ablation experiments

7.1 Template influence

In order to investigate the effect of adopting different templates for the proposed model, we select two templates (No.1 and No.2) which are manually designed by Cui et al. [10], namely "<candidate_span> is a <entity_type> entity." and "The entity type of <candidate_span> is <entity_type>.",

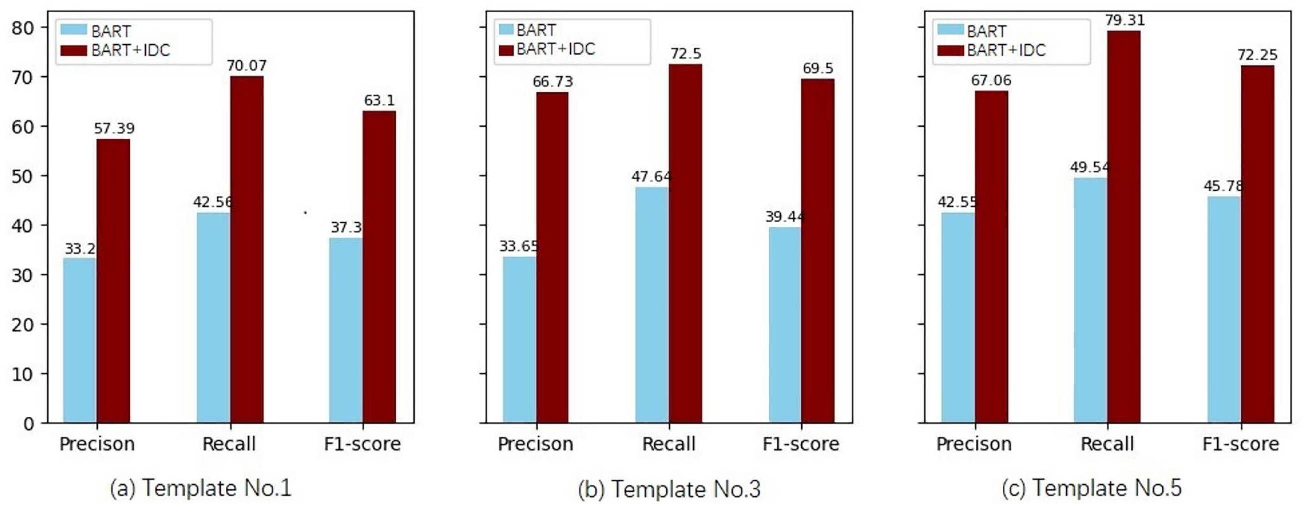

Fig. 5 Evaluation of the interaction detection module

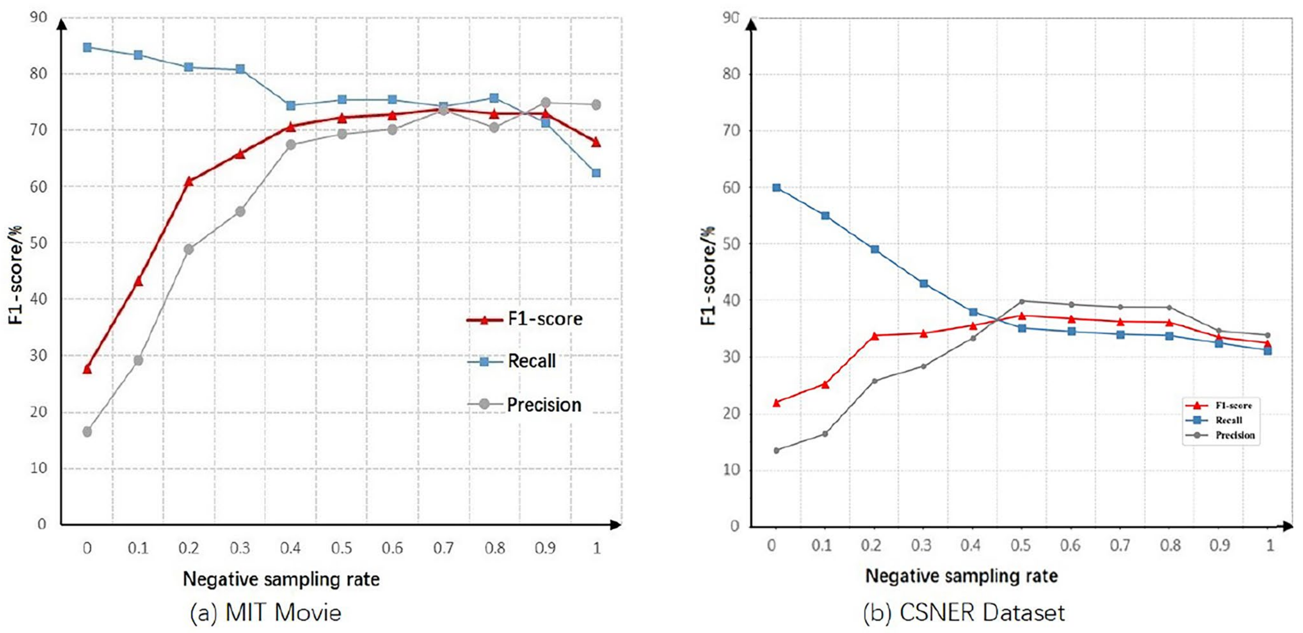

Fig. 6 Metrics changing trend with the negative sampling rate

and the top 3 templates (No.3–5) which are automatically generated by our model, as shown in Table 2, to conduct the comparison experiments on 10-shot MIT Movie dataset. The experimental results of the 5 different templates are shown in Table 9.

As can be seen from Table 9, template No.5 which are automatically generated achieves the best performance in terms of precision, recall and F1 score. In the experiments conducted by Cui et al. [10], the CoNLL03 dataset was used to evaluate the influence of different templates, and template No. 1 performs better than template No. 2, while in our experiment, template No. 2 performs better than template No. 1., which indicates that manually designed templates are not robust to different datasets. Template No. 5 shows better performance than template No. 1 on both the CoNLL03 and MIT Movie dataset, indicating that the automatically generated template is more robust. During the template generation phase, the template scores of No. 3–5 range from low to high, and the F1 scores are also ranked accordingly, indicating the reliability of the automatically generated templates. From above we can see that manually designed templates are sensitive to different datasets, and huge efforts are required to evaluate and prove the reliability of the manual templates, while automatically generated templates are more robust, the proposed model can directly generate the optimal templates and then applied to NER task.

7.2 Interaction detection module

In order to verify the effectiveness of the interaction detection module, we choose the BART model as the baseline, and conduct three comparison experiments on the 10-shot MIT Movie dataset. We use three different templates including template No. 1, templates No. 3 and No. 5 as shown in Table 9 (including manual templates and automatic templates) to construct prompts, and send to BART model, and BART integrating with interaction detection module (abbreviated as BART+IDC) for prediction, the experimental results are shown in Fig. 5. As can be seen from Fig. 5, BART integrating with interaction detection module (BART+IDC) significantly outperforms the BART model in terms of precision, recall, and F1 score, which validates that the interaction detection module can improve the performance of entity recognition.

7.3 Influence of negative sampling rate

To evaluate the influence of negative sampling rate α on the performance of the model, we conduct a series of experiments with an interval of 0.1 on the 10-shot MIT Movie and CSNER datasets, and the experimental results are shown in Fig. 6. As can be seen from Fig. 6, on MIT Movie dataset, the precision gradually increases while the recall keeps decreasing as the negative sampling rate α becomes larger, the F1 score first increases and then decreases. When α=0, i.e., all the non-entity X_{prompt} are discarded in the training phase, the model is more inclined to predict that the input text contains named entities, so the recall reaches 84.62%, but the precision is only 16.57%. When α=1, i.e., all non-entity X_{prompt} are retained in the training phase, in this case there are too many negative samples, the model is more inclined to predict that the input text does not contain any named entity, so the recall is the lowest, only 62.36%, the precision reaches 74.47%. The similar trend is observed on the CSNER dataset. When α is in the range of [0.5, 0.8], the number of entity samples and non-entity samples are more balanced, the model predicts more accurately and holds the highest F1 score.

8 Discussion

8.1 Complexity

The time complexity of the proposed model is mainly concentrated in the prompt construction and learning. Assuming that the number of entity types is *k* and the length of the input text is *l*, then the computational time complexity of the proposed method is $O(k \cdot l^2 d)$, where *d* is the vector dimension of the language model. Compared to TemplateNER [10] which used sliding windows to construct prompts, which first enumerate all the candidate spans in the sentence and then fill them in the handcrafted templates, its time complexity is $O(lk\hat{n} \cdot l^2 d)$, where $\hat{n}$ is the maximum length of the sliding window n-grams. It can be seen that our method avoids enumerating spans and significantly reduces the computational complexity.

To provide an intuitive understanding of the model complexity, we compare our method with TemplateNER in terms of training/inference time on the CoNLL03 dataset. The training/inference time is sensitive to the experimental environment, including the number of training samples, batch size and the total number of epochs. Our model demonstrates comparable training efficiency to TemplateNER, with a total training time of 72 min, compared to 80 min for TemplateNER. In terms of inference speed, our method is significantly faster: the inference time is 12 min, whereas TemplateNER requires 46 min. All timing measurements were obtained using a single RTX 4090 24G GPU, with a batch size of 32 and 40 training epochs.

8.2 Limitations

Although the proposed model consistently outperforms baseline models, it is important to recognize and address its inherent limitations.

- *Computational Limitation*. We construct prompts for each entity type and then perform entity locating, which requires multiple rounds of prompting. For an input with *k* entity types, the prompt learning requires *k* predictions. The prompt construction strategy introduces a computational overhead that grows with the number of entity types. One potential solution is to design a unified prompting strategy that can handle multiple entity types simultaneously in a single round.
- *Application Scenarios Limitation*. The proposed model is primarily designed for flat NER tasks. In scenarios where entities are nested within others, its performance tends to degrade due to its lack of structural mechanisms. How to leverage entity boundary information to handle nested NER needs further research.
- *Domain Limitation*. The training corpus for the pre-trained model is mainly sourced from general domains, creating a significant domain gap when applied to highly specialized domains, which may affect its performance. To further enhance the effectiveness in highly specialized domains, we can apply warmup training before

employing the proposed model in few-shot NER. During the warmup training stage, we can extract domain-specific data from rich and easily accessible sources and design simplified tasks to perform warmup training.

8.3 LLMs in few-shot NER

Large Language Models (LLMs), such as GPT-4 [33] and LlaMA [34], exhibit significant effectiveness in various NLP tasks due to their extensive training scale and numerous model parameters. However, recent studies have shown that existing large models still struggle to achieve satisfactory performance on NER tasks. For instance, the GPT−3.5 series models achieve a micro-F1 score ranging from only 20 to 60 points on NER task [35]. Even the popular ChatGPT can only achieve F1 scores of 67.2% and 51.1% for NER on CoNNL2003 and OntoNotes respectively [36]. The performance can be enhanced through careful prompt engineering and instruction-tuning techniques, but these refined LLMs can hardly outperform discriminative encoder models such as BERT and RoBERTa [37]. Looking ahead, the potential of LLMs to enhance NER should not be underestimated, promising directions to enhance LLM adaptability for NER including prompting, fine-tuning and hybrid approaches that combine LLMs with specialized NER need further research [38].

Although LLMs excel in various NLP tasks, our approach provides unique advantages in efficiency, adaptability, and customization for specialized NER tasks. First, our model is lightweight and efficient to train, in contrast to LLM's model size with billions of parameters. Second, our model demonstrates adaptability to highly specialized domains, consistently achieving impressive results. Third, our model is more extensible and flexible than LLMs, enabling easy modification to enhance its performance.

9 Conclusion and future work

In this paper, we investigate an automatic prompting framework for few-shot NER. In contrast to previous prompt-based methods, we introduce a knowledge enhanced template generation method to automatically generate the optimal templates, and the experimental results show these templates outperform manual templates and require less human effort. Next, we construct prompts for each entity type and feed them into the MLM for entity prediction, which do not need to enumerate all the possible spans, thus reducing the computational cost. To obtain accurate entity prediction results, we also design an interaction detection module to improve the entity recognition performance. Experimental results show that our model outperforms the existing competitive models on both rich-resource settings and few-shot settings. For future work, we will focus on reducing the complexity and integrating LLMs into NER tasks.

Acknowledgements The research is supported in part by the National Natural Science Foundation (62102310, 62102324), Xixian New Area Science and Technology Plan Project (RGZN-2023-002, 2022-ZDJS-001), the China Postdoctoral Science Foundation (2020M683689XB), the Youth Innovation Team of Shaanxi Universities (2019-38), and the Project of Zigong Science and Technology Bureau (2022XDHZ-XAJD-01).

Author contributions ZI: Methodology, Experiment, writing—Original draft. TQ: Methodology, Writing- review and editing, Supervision. BL: Data curation, Experiment, Validation. QS: Writing-review and editing. SL: Validation

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no Conflict of interest.

Ethical and informed consent for data used This paper was done by the authors, and no human participants other than the authors were involved in it, and informed consent was obtained from all authors.

References

1. Li J, Sun A, Han J, Li C (2020) A survey on deep learning for named entity recognition. IEEE Trans Knowl Data Eng 34(1):50–70
2. Chiu JP, Nichols E (2016) Named entity recognition with bidirectional lstm-cnns. Trans Assoc Comput Linguist 4:357–370
3. Peters ME, Ammar W, Bhagavatula C, Power R (2017) Semi-supervised sequence tagging with bidirectional language models. Preprint at arXiv:1705.00108
4. Huang J, Li C, Subudhi K, Jose D, Balakrishnan S, Chen W, Peng B, Gao J, Han J (2020) Few-shot named entity recognition: a comprehensive study. Preprint at arXiv:2012.14978
5. Snell J, Swersky K, Zemel R (2017) Prototypical networks for few-shot learning. In: Proceedings of the 31st international conference on neural information processing systems, pp 4080–4090
6. Fritzler A, Logacheva V, Kretov M (2019) Few-shot classification in named entity recognition task. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing, pp 993–1000
7. Ma T, Jiang H, Wu Q, Zhao T, Lin C-Y (2022) Decomposed meta-learning for few-shot named entity recognition. Preprint at arXiv:2204.05751
8. Liu AT, Xiao W, Zhu H, Zhang D, Li S-W, Arnold A (2022) Qaner: Prompting question answering models for few-shot named entity recognition. Preprint at arXiv:2203.01543
9. Chen X, Li L, Deng S, Tan C, Xu C, Huang F, Si L, Chen H, Zhang N (2021) Lightner: a lightweight tuning paradigm for low-resource ner via pluggable prompting. Preprint at arXiv:2109.00720
10. Cui L, Wu Y, Liu J, Yang S, Zhang Y (2021) Template-based named entity recognition using bart. In: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pp 1835–1845

11. Jiang Z, Xu FF, Araki J, Neubig G (2020) How can we know what language models know? Trans Assoc Comput Linguist 8:423–438
12. Radford A, Narasimhan K, Salimans T, Sutskever I (2018) Improving language understanding by generative pre-training
13. Kenton JDM-WC, Toutanova LK (2019) Bert: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT 2019, vol 1, pp 4171–4186
14. Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L (2020) Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Proceedings of the 58th annual meeting of the association for computational linguistics, pp 7871–7880
15. Liu M, Tu Z, Zhang T, Su T, Xu X, Wang Z (2022) Ltp: a new active learning strategy for crf-based named entity recognition. Neural Process Lett 54(3):2433–2454
16. Liu Y, Wei S, Huang H, Lai Q, Li M, Guan L (2023) Naming entity recognition of citrus pests and diseases based on the bert-bilstm-crf model. Exp Syst Appl 234:121103
17. Hou Y, Che W, Lai Y, Zhou Z, Liu Y, Liu H, Liu T (2020) Few-shot slot tagging with collapsed dependency transfer and label-enhanced task-adaptive projection network. Preprint at arXiv:2006.05702
18. Yang Y, Katiyar A (2020) Simple and effective few-shot named entity recognition with structured nearest neighbor learning. In: Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP), pp 6365–6375
19. Das SSS, Katiyar A, Passonneau RJ, Zhang R (2021) Container: few-shot named entity recognition via contrastive learning. Preprint at arXiv:2109.07589
20. Wang H, Xu C, McAuley J (2022) Automatic multi-label prompting: simple and interpretable few-shot classification. Preprint at arXiv:2204.06305
21. Zhu Y, Wang Y, Qiang J, Wu X (2023) Prompt-learning for short text classification. IEEE Trans Knowl Data Eng 36(10):5328–5339
22. Zhu Y, Wang Y, Mu J, Li Y, Qiang J, Yuan Y, Wu X (2024) Short text classification with soft knowledgeable prompt-tuning. Expert Syst Appl 246:123248
23. Chen Z, Li Z, Zeng Y, Zhang C, Ma H (2024) Gap: a novel generative context-aware prompt-tuning method for relation extraction. Exp Syst Appl 248:123478
24. Li X, Yuan S, Gu X, Chen Y, Shen B (2024) Few-shot code translation via task-adapted prompt learning. J Syst Softw 212:112002
25. He Y, Huang X, Zou S, Zhang C (2024) Psan: prompt semantic augmented network for aspect-based sentiment analysis. Exp Syst Appl 238:121632
26. Ma R, Zhou X, Gui T, Tan Y, Li L, Zhang Q, Huang X (2021) Template-free prompt tuning for few-shot ner, 5721–5732
27. Ye F, Huang L, Liang S, Chi K (2023) Decomposed two-stage prompt learning for few-shot named entity recognition. Information 14(5):262
28. Shen Y, Tan Z, Wu S, Zhang W, Zhang R, Xi Y, Lu W, Zhuang Y (2023) Promptner: prompt locating and typing for named entity recognition. arXiv preprint arXiv:2305.17104
29. Huang J, Yan D, Cai Y (2024) Pmrc: Prompt-based machine reading comprehension for few-shot named entity recognition. In: Proceedings of the AAAI conference on artificial intelligence, vol 38, pp 18316–18326
30. Sang ETK, De Meulder F (2003) Introduction to the conll-2003 shared task: Language-independent named entity recognition. In: Proceedings of the seventh conference on natural language learning at HLT-NAACL 2003, pp 142–147
31. Liu J, Pasupat P, Cyphers S, Glass J (2013) Asgard: a portable architecture for multilingual dialogue systems. In: 2013 IEEE International conference on acoustics, speech and signal processing, pp 8386–8390
32. Yan H, Gui T, Dai J, Guo Q, Zhang Z, Qiu X (2021) A unified generative framework for various ner subtasks. In: Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers), pp 5808–5822
33. Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, Almeida D, Altenschmidt J, Altman S, Anadkat S et al (2023) Gpt-4 technical report. arXiv preprint arXiv:2303.08774
34. Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M-A, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F et al (2023) Llama: open and efficient foundation language models. arXiv preprint arXiv:2302.13971
35. Chen X, Ye J, Zu C, Xu N, Zheng R, Peng M, Zhou J, Gui T, Zhang Q, Huang X (2023) How robust is gpt-3.5 to predecessors? A comprehensive study on language understanding tasks. arXiv preprint arXiv:2303.00293
36. Li B, Fang G, Yang Y, Wang Q, Ye W, Zhao W, Zhang S (2023) Evaluating chatgpt's information extraction capabilities: an assessment of performance, explainability, calibration, and faithfulness. arXiv preprint arXiv:2304.11633
37. Li Z, Li X, Liu Y, Xie H, Li J, Wang F-L, Li Q, Zhong X (2023) Label supervised llama finetuning. arXiv preprint arXiv:2310.01208
38. Keraghel I, Morbieu S, Nadif M (2024) Recent advances in named entity recognition: a comprehensive survey and comparative study. arXiv preprint arXiv:2401.10825