Analysis of Embedded System's Functional Requirement using BERT-based Name Entity Recognition for Extracting IO Entities

Man Yiu Chow[1,a)]

Received: March 25, 2022, Revised: July 8, 2022,

Accepted: September 20, 2022

Abstract: The functional testing specification is usually designed based on the IO entities' recognition from the embedded system's functional requirement sentences. However, it is hard for the software testing engineers to ably recognize the appropriate IO entities from the functional requirement sentences without clearly indicated entities and much experience with the domain knowledge. The conventional rule-based methods of extracting IO entities are inapplicable when the requirement sentences drafted by humans become too semantically complex. Even though all the sentences keep aligned with the structure, it is still infeasible to manually hard-code each rule when those rules change from time to time without any explicit writing standard. With the successful application of artificial intelligent techniques in natural language processing (NLP), we propose a method that intelligently solves the issue of the entities recognition by using BERT (Bidirectional Encoder Representations from Transformers) based named entity recognition (NER) which is the technique of NLP to recognize the phrases having similar attributes in semantics. In this paper, we specifically focus on the issue of IO entities' recognition in the embedded systems that implement the inverter control function such as elevator and hybrid hydraulic excavator systems. Our evaluation result demonstrates that the best model variant fine-tuned on 829 sentences achieves more than 80% F-measure in recognizing the IO entities, and the model can provide applicable information for the improvement of industrial productivity in the target industries. Our contribution of this paper is to provide insight into the case whether the IO entities in the target system manages to be interpreted well by exploiting the BERT model with the sole reliance on the small size of exemplary IO entities data and three existing model variants pre-trained on large corpus open datasets with general language knowledge.

Keywords: software requirement, named entity recognition, BERT, NLP, IO entities

1. Introduction

To ensure high software operation quality of embedded systems, especially of which implements the inverter control function such as elevator and hybrid hydraulic excavators system, the functional requirement sentences of those software designs are always provided by different software developers such that the software testing engineer can verify the corresponding software codes and designs for their performance satisfaction based on the sentences of those given requirements to maintain a good quality of the software architecture and operation. However, one of the big challenges for software testing engineers is to recognize the target IO entities accurately and swiftly from the functional requirement sentences with different writing styles in natural language. Those entities are usually not clearly written, and it requires intelligent semantic analysis for the IO entities' recognition.

Even though there are other researches on crafting the standard template of writing elicit requirements such as Easy approach to requirements syntax (EARs) [1] and specific representation template [2] to improve the efficiency and accuracy of those functional requirement sentences, different software developers could still have changed writing standard to adapt the complexity of some exceptional requirement sentences and might not be able to follow the design writing rules such as "If...Then...". From the perspective of software developers, the conciseness and semantics of the sentences are far more important than the compliance and the quality of those standard writing formats when they solely consider efficient communication with software testing engineers.

Due to the varying writing styles, the software testing engineers might count on their experience but not simply sentence

1 Service Systems Innovation Center – DX Engineering Research Department, Hitachi Ltd. Research and Development Group, Yokohama, Kanagawa 244–0817, Japan

a) manyiu.chow.dv@hitachi.com

structure to pinpoint the target IO entities correctly during the analysis of the software functional requirements. Their experiences are not usually standardized as the decisions of recognizing the target IO entities from software testing engineers are slightly different from each other due to their divisive working backgrounds. This further takes more costly man-hours to ensure the quality of the reviewing process. For this reason, it is more practical to exploit an automation program for making swift, reliable, and uniform recognition based on the underlying design rules for better consistency to assist software testing engineers while extracting the IO entities from natural language's functional requirement sentences. However, it is impossible to create such kind of automation program merely on a basis of a staggering number of rules from diverse templates (E.g., EARs) as well as exceptional cases due to the special wordings and writing standards as the rule-based method could barely understand the complicated IO entities' relation based on the prerequisite domain knowledge in a semantic way.

With the existence of more advanced Artificial intelligent techniques and its past successful applications in the Natural language processing (NLP) field, we decided to utilize deep learning based NER (Named Entities Recognition) to intelligently extract target IO entities in the natural language form functional requirement sentence when its deep learning-based technique has been recently proven for the exceptional abilities to understand natural language and complex relation in the sentences. In general, NER is one of those prominent NLP methods to extract a phrase that clearly identifies one item from a set of other items in a sentence that have similar attributes. [3] There are several approaches to achieve NER in the past including dictionary-based NER and conventional machine learning-based NER. [4] The former one requires an exhaustive dictionary list and is easily broken when a new word or unseen structure does not match any item on the prepared list. The latter one such as SVM (Support vector machine) CRF (conditional random field) [5] is only able to deal with the common cases with simple structure learned from training data as those methods only manage to capture sallow dimension features. For this reason, we only focus on the contemporary deep learning-based NER in this paper.

Choosing the contemporary deep learning-based NER with the lack of a large corpus of embedded system's technical sentences for training model, we experimented on three common variants of the state-of-the-art BERT model pre-trained with public open datasets which consist of the data from BookCorpus with 800 million words and the data from English Wikipedia with 2500 million words. [6] We chose the BERT model because it has been demonstrated to attain more than 90% F-measure performance in both CoNLL03 and OntoNotes5.0 benchmark for NER. [5] In other words, our paper contributes the insight whether the three target variants of the BERT model pre-trained with the data of general English knowledge can be fine-tuned with a small size of dataset related to the functional requirement of an embedded system implemented with inverter control function and therefore let the fine-tuned models provide accurate prediction of IO entities' recognition for consistent embedded system's functional requirement analysis.

2. Problem Formulation

2.1 The Definition of Target IO Entities for the Analysis

In the IO entities of our functional requirement sentences, we mainly target 3 types of system's IO entities, "input", "output", and "condition" in the requirement. The overview of the relation of these three components in the embedded system's functional requirement is illustrated in the following diagram.

In the requirement shown in **Figure 1**, all the components have dependent relations in which the input component proceeds to the output component through passing the condition component. When the figure elucidates all the components in the requirement, those components become indirect and indistinct in natural language sentences.

Due to such characteristics in natural language sentences, a rule-based method is therefore incapable of capturing such relations because there is no straightforward rule by relying on standard words or specific grammar structures to extract the relations. Without the standard writing template, the rule-based extraction method is also unevaluable and unachievable as the developer requires to manually address thousands of exceptional cases and writing styles.

Even though a deep learning-based NER model can automatically figure out the IO entities from those natural language sentences for us, the manual preparation of the training dataset is still essential ahead of inference to guide the model for molding the above underlying semantic rules. To capture those IO entities along with those relations in the requirement sentences for the annotation of training data, the problem of setting up clear definitions of each component is significant to figure out the proper annotation rule in a bid to ensure good labelling consistency and

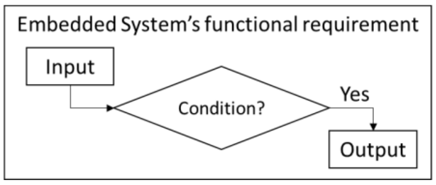

Fig. 1 The overview of the relation of these three components in embedded system's functional requirement.

good model generalization during the model training, and we addressed this issue in section 4 of this paper.

2.2 The Selection of NLP Model for NER Task

To select the most appropriate NLP model for the NER task, we considered two factors. The first one is the ability of good model generalization without the preparation of overwhelmingly large training datasets when it is expensive to manually label the data. The second factor is the capability of learning complicated relations for each entity when the relations are abstract in semantics because the relations of IO entities are not always observable by relying on the simple sentence structure. Especially in our case, the optimal NER method should be able to recognize the underlying semantics relations of inputs, outputs and passing conditions based on the software functional requirements even if the writing style and sentence structure change on a case-by-case basis. This implies that our model should be able to learn the abstract concept of the IO entities' relations from the general English knowledge and our limited size of available training dataset due to the insufficient data variety from our products.

Conventionally, most researchers use CRF-based NER methods which are the popular machine learning-based NER to achieve NER tasks. Those methods have been successfully applied to different applications such as biomedical text, tweets, and chemical text. [4] Despite its popularity and successes, solely relying on original CRF based methods without additional modification to the model structure are still unable to capture the underlying complicated features with only little available training datasets due to the lack of the efficient ability to deal with unknown tokens and the poor support of transfer learning from other trained models. [7], [8]

For these reasons, a deep learning-based NER model is focused on in our paper when it was proven to be able to capture underlying features and to support transfer learning. Among deep learning-based NER models, transformer-based BERT (Bidirectional Encoder Representations from Transformers) has been proven to achieve state-of-the-art performance [7], [9] compared to other types of neural network models in not only NER task but also different NLP tasks including SQuAD2.0 (Question answering). [6]

Our problems in this paper are how different variants of BERT models perform and how we handle the fine-tuning with our limited size of available training dataset for the application of IO entities' recognition in our functional requirement's analysis. The details of the model as our solution are elaborated in section 3 of this paper.

3. BERT based NER for Functional Requirements

This section presents the theory of BERT based NER and the details of solving the recognition of target IO entities in the embedded system's functional requirements using BERT. In section 3.1, the details of the BERT structure and the variant of its pre-trained model are illustrated for the significance of solving the recognition problem. In section 3.2, The fine-tuning step for the NER task and its additional BERT structure are explained in detail.

3.1 BERT Model and its Variants of Pre-trained Model

The base BERT exploits the concept of transformer which highlights the reliance on attention mechanism to generalize language knowledge well in most of the NLP tasks without the need for recurrence and convolutions. [10] The attention mechanism succeeds when it manages to efficiently learn long-range dependencies faster due to the shorter paths between the input and output sequences compared to other types of neural network layers such as convolutional or recurrent layers. In BERT, it only takes the transformer encoder part which has the bidirectional self-attention structure, and the part is originally for understanding the language before feeding into the decoder part in the whole transformer model.

As shown in **figure 2**, the model consists of 12 layers transformer block with 768 hidden sizes and 12 bidirectional self-attention heads, and the number of the total parameters is around 110 million. [6] To enable the model to understand the sequential order of tokens, the tokens input will be processed into three embeddings inside the first representation layer of BERT before the transformer encoder. The three embeddings include the corresponding token for the target words, the segment for the differentiation of the belonging between two input sentences based on the token [SEP], and position embeddings for sequential order of words in a sentence.

To acquire the general language representation, the BERT

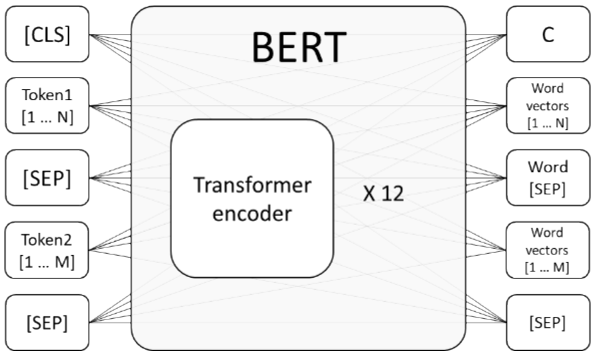

Fig. 2 The BERT model during pre-training step.

model has to be pre-trained with a large language corpus to learn the target language's features. There are two essential tasks in the pre-training procedure, the Masked language model (MLM) and Next sentence prediction (NSP). [6] For the input of MLM during pre-training, some of the tokens in the sentences are masked with the token [MASK] by random and then the model will be trained by feeding the original word vectors output in the masked token position. For the input of NSP, two sets of tokens, {Token1_1,...,Token1_N} and {Token2_1,...,Token2_M}, are formed respectively from two sentences and separated by the token [SEP]. The model will thereby be trained by feeding the first output word C to indicate whether those two sentences are relevant to each other.

In the target pre-trained models for functional requirement analysis in this paper, we chose three common variants and specifically targeted (1) Cased BASE model which is pre-trained with case-sensitive corpus, (2) Uncased BASE model which is pre-trained with lowercased corpus and (3) MNLI (Multi-Genre Natural Language) which is further fine-tuned with The Multi-Genre Natural Language Inference dataset [11] on the uncased BASE model. For the corpus mentioned above, we follow the pre-training data from the original paper [6] by using BookCorpus with 800 million words and English Wikipedia with 2500 million words. The significant difference between these models is the way to handle the pre-training process. Since the performance of the fine-tuned NER models varies due to different pre-trained model, the paper provides an insight whether the BERT model pre-trained on general English corpus is suitable and which variant of pretrained model is the most suitable for the analysis of functional requirement in embedded system with many different technical terminologies in case the large corpus of technical document training data is not available.

3.2 Fine-tuning BERT to NER Task

After pre-training the BERT to acquire general language knowledge, the model is then prepared to be fine-tuned to conduct NER task. To turn the pre-trained model into the NER model, the last layer of the BERT is converted to a token classifier which is shown in **Figure 3** below.

Specifically, we treat all the input tokens as if they are the words from a single sentence and there is no longer any token [SEP] to separate two sentences. In other words, our input can be more than one sentence but there is no special delimiter to separate those sentences.

For each corresponding output node $Label_i$ of the model, it is responsible for the label classification of the token $Token_i$ in the corresponding order. Depending on the number of label types, the size of the label vector in each output node would be different. In the paper, we adopted BIO (Beginning, inside, and outside) labelling scheme when the scheme is conventionally popular for the NER labelling task, each label vector $Label_i$ in the node for $Token_i$ has 8 classifications including "O" for tokens without any label, "B-input" and "I-input" for input label, "B-output" and "I-output for output label, "B-condition" and "I-condition" for condition label and "PAD" for padding tokens which are empty.

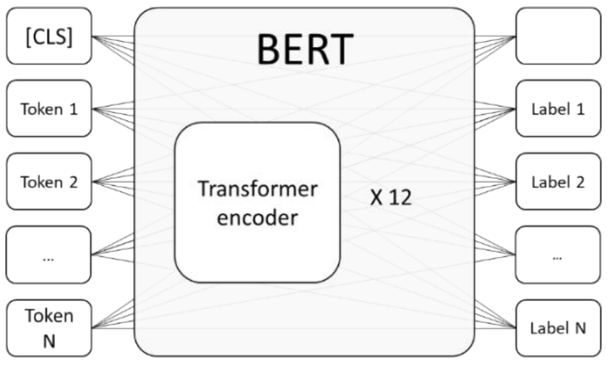

Fig. 3 The BERT model during the fine-tuning step for NER task.

4. Model Fine-tuning and Dataset Preparations

In this section, the fine-tuning dataset preparation with the clear definition of the three target components for functional requirements and the implementation of model fine-tuning in the IO entities' recognition for functional requirements are detailed in section 4.1 and section 4.2 respectively.

4.1 Fine-tuning Dataset Preparation

To prepare fine-tuning dataset, we annotated our data for the AI model to understand the semantic feature of each target label for the IO entities in functional requirement. Meanwhile, we have set up the annotation rules to address the following three data quality issues during the annotation. Addressing the issues helps us achieve higher accuracy especially when only a few data are available.

The first data quality issue is the consistency of the annotation methods from different annotators as we should avoid any double standard for the definitions of the same labels. For example, the consideration of whether the output element should include the action verb is important for the model to accurately recognize the coverage of the output element in any condition. Supposed that action verb should be treated as an output element, the model could hardly converge to such annotating way if some of the annotated sentences do not follow the same rule. In other words, high semantic consistency for the annotation can provide good model generalization and vice versa. In the paper, the definitions of all target components have been listed in the **Table 1**.

Table 1 The definition of each label.

Labels	Definitions
Input	The operation parameter or variable which is required for the system to conduct the output such as time and.
Output	The products or the actions which will be conducted after processing the input and addressing the condition requirement.
Condition	The rules to pass before triggering the output

The second data quality issue is the undesirable training data noise, and the annotation rule should help annotators spot them out for data cleansing. In our preparation, we specifically remove the sentences under drafting such as the sentence with only the word "TBD" or the sentence with only a website address or a file path available. Additionally, we also removed the overwhelming long sentence with undetermined relations due to the special project rule. Training the model with those undesirable sentences usually results in lower detection accuracy because those data are usually sparse with only the specific type of label in the whole sentence, and it evens out the trained model weights.

The third data quality issue is to diversify our dataset with different sentence structures to capture the general semantics feature of each target component instead of targeting the positions of the words based on the specific keyword or sentence structure. For example, X components in these 2 sentences "... by using X" and "... while using X" could be input and condition respectively.

In our dataset, we have annotated 873 records in total with "input" labels for 3964 tokens, "output" labels for 14985 tokens, and "condition" labels for 9927 tokens. The 829 records are for model fine-tuning and the rest of the 44 records are for performance evaluation.

4.2 The Implementation of Model Fine-tuning

Regarding fine-tuning model configuration, we trained three models with Adam optimizer with weight decay for 10 epochs. According to the fine-tuning procedure in the original BERT model, the author conducted 3 epochs of fine-tuning on each of The General Language Understanding Evaluation (GLUE) datasets respective. Each dataset has more than 2000 records. Since we have only around 800 fine-tuning records, the fine-tuning epoch increase to avoid insufficient converging to optimal loss. For the rest of the hyperparameters, we selected 5e-5 as our learning rate and 2 as our batch size. We chose a smaller batch size as a small batch size tends to have the effect of stochastic gradient descent and achieve better model regularization.

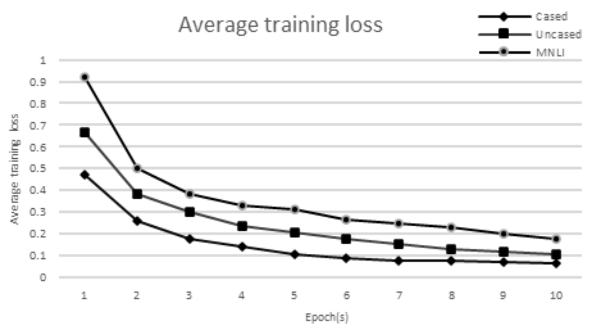

Fig. 4 The average training loss of three model variants for 10 epochs.

Figure 4 shows the average training loss of 3 targeted model variants, the Cased BERT model, the Uncased BERT model and the MNLI BERT model. Among these three models, the Cased BERT model suffers the least average training loss after 10 epochs.

For the input token of the fine-tuned NER model, the maximum token for each sentence is 100. In other words, the model will pad the sentence with the token "PAD" if it has less than 100 tokens and the redundant tokens will also be discarded when the sentence has more than 100 tokens.

5. Performance Evaluation

5.1 Evaluation Method

In the paper, we adopted the F1-measure shown in the following formula as our performance indicator for each label. It reflects a harmonic mean of precision and recall. Our target label includes the "Input" label, the "Output" label, the "Condition" label and "O" for a token without any label.

$$F1 - measure = \frac{2 * Precision * Recall}{Precision + Recall} \qquad (1)$$

It is used to measure a model's prediction accuracy because it is not comprehensive to focus only on either precision or recall. There is always a prediction problem if only one of them achieves a higher value.

5.2 Evaluation Environment

To build BERT-based NER. we created our BERT model using HuggingFace library [12] with a fully connected layer on top for token classification. For the three variants of pre-trained BERT models, they were pretrained and provided by the open-source community since pre-training model with a large corpus requires costly computational resources. In our experiment, we conducted fine-tuning on three variants of BERT models.

Our fine-tuning step was carried out on the server with NVIDIA V100 32 GB GPU memory and 64 GB RAM and we did not run out of memory during fine-tuning. Our dataset has 873 records in total with "input" labels for 3964 tokens, "output" labels for 14985 tokens, and "condition" labels for 9927 tokens. We fine-tunned the BERT model with 2 batch sizes and Adam optimizer with weight decay for 10 epochs using 829 data records in total and the rest of the 44 data records are for performance evaluation.

5.3 Results

In our evaluation, we have tested 44 records that contain 95 functional requirement sentences with 2257 tokens in total.

In our result of the Cased BERT model illustrated in **Table 2**, it shows that all the labels achieve more than 80% F1-measure. The "Output" label achieves the highest F1-measure score with 92% whereas the "Input" label achieves the lowest F1-measure score with 82% when the precision is relatively low with 74% compared to the rest of the labels.

In our result of the Uncased BERT model illustrated in **Table 3**, it shows that all the labels achieve more than 80% F1-measure. The "Output" label achieves the highest F1-measure score with 93% whereas the "Input" label achieves the lowest F1-measure score with 81% when the precision is relatively low with 76% compared to the rest of the labels. The overall performance of this model is similar to that of the cased BERT model.

In our result of the MNLI BERT model illustrated in **Table 4**, it shows that most of the labels achieve around 70% F1-measure. The "Output" label achieves the highest F1-measure score with 86% whereas the "Input" label achieves the lowest F1-measure score with 65% when the recall is relatively low with 62% compared to the rest of the labels. The overall performance of this model variant is worse than the rest of the other BERT model variants.

From the comparison of three variants of pre-trained model shown in **Figure 5**, the MNLI model suffers the worst performance among the three variants whereas the performance of the rest of the two models is similarly good with more than 80% average F1-measure for all labels. Furthermore, we found that "output" labels are well recognized, but this is not the case for "input" labels in all three models since most of the common functional requirements in our fine-tuning dataset contain "output" elements with 14985 labels in total, but this is not the case for "input" elements with 3964 labels in total.

Table 2 Cased BERT model.

Labels	Precision	Recall	F1-measure
Input	0.74	0.92	0.82
Output	0.97	0.88	0.92
Condition	0.87	0.96	0.91
O	0.90	0.88	0.89

Table 3 Uncased BERT model.

Labels	Precision	Recall	F1-measure
Input	0.76	0.88	0.81
Output	0.96	0.90	0.93
Condition	0.83	0.95	0.88
O	0.96	0.88	0.91

Table 4 MNLI BERT model.

Labels	Precision	Recall	F1-measure
Input	0.69	0.62	0.65
Output	0.86	0.86	0.86
Condition	0.75	0.84	0.79
O	0.77	0.67	0.72

5.4 Exemplary Results and Analysis

The 6 representative examples are illustrated to explain the component recognition ability of three models and all those sentences were chosen based on the common structures in most testing data sentences. In the prediction results of our examples, we highlighted "input" labels in pale green, "output" labels in pale orange and "condition" labels in yellow.

The functional requirement sentence in example 1 contains the output component and the condition component. Both the Cased model and the Uncased model are able to recognize all the components whereas the MNLI model fails to target the action words "determinate" as an "output" token.

The functional requirement sentence in example 2 contains

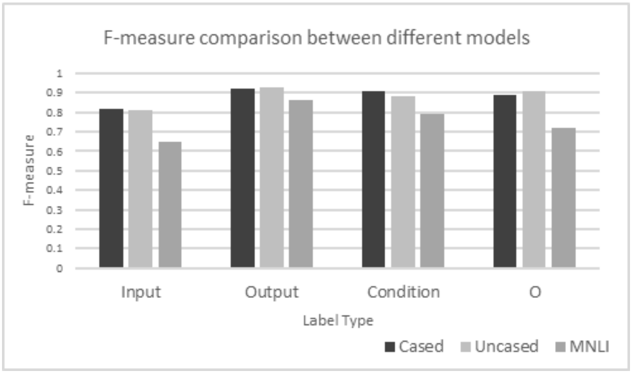

Fig. 5 F-measure comparison between three models.

Table 5 Example 1.

Models	Sentence and labels
Ground truth	[Software module name] determinate hw type only once, when [software system name] start.
Cased model	[Software module name] determinate hw type only once, when [software system name] start.
Uncased model	[Software module name] determinate hw type only once, when [software system name] start.
MNLI model	[Software module name] determinate hw type only once, when [software system name] start.

Table 6 Example 2.

Models	Sentence and labels
Ground truth	[Software module name] shall execute [Specific function name for current detection] in pwm cycle.
Cased model	[Software module name] shall execute [Specific function name for current detection] in pwm cycle.
Uncased model	[Software module name] shall execute [Specific function name for current detection] in pwm cycle.
MNLI model	[Software module name] shall execute [Specific function name for current detection] in pwm cycle.

Table 7 Example 3.

Models	Sentence and labels
Ground truth	[Software module name] shall calculate 3-phase voltage commands from compensated theta θcmp, sine (θ), cosine (θ) and, d / q - axis voltage commands
Cased model	[Software module name] shall calculate 3-phase voltage commands from compensated theta θcmp, sine (θ), cosine (θ) and, d / q - axis voltage commands
Uncased model	[Software module name] shall calculate 3-phase voltage commands from compensated theta θcmp, sine (θ), cosine (θ) and, d / q - axis voltage commands
MNLI model	[Software module name] shall calculate 3-phase voltage commands from compensated theta θcmp, sine (θ), cosine (θ) and, d / q - axis voltage commands

output component and input component. Both the Cased model and the Uncased model are able to recognize all the components whereas the MNLI model fails to target the time cycle as an "input" token.

The functional requirement sentence in example 3 contains the output component and the input component. All the models can recognize all the components. This example demonstrates the model's ability to understand the target input and the target output based on the semantics of the sentence.

The functional requirement sentence in example 4 contains condition component, output component and input component. All the models managed to recognize all the components. The complicated condition pattern in the example can be captured successfully by the models and it also shows that the models can recognize the long output sentence.

The functional requirement sentence in example 5 contains the output component and the input component. All the models managed to recognize all the components. Similar to example 4, the complicated output pattern extended in the example can be captured successfully by the models and it also shows that the models can recognize the long output sentence.

The functional requirement sentence in example 6 contains the output component and the input component. Cased and Uncased models are able to recognize all the components whereas

Table 8 Example 4.

Models	Sentence and labels
Ground truth	[Software module name] shall judge the execution of following diagnosis depends on following precondition in specified cycle. Diagnosis: Functional restriction because of system power supply low Precondition: Inverter system shall perform this diagnosis when "System state"!= [Specific system mode name]. Cycle: [specific processing cycle]
Cased model	[Software module name] shall judge the execution of following diagnosis depends on following precondition in specified cycle. Diagnosis: Functional restriction because of system power supply low Precondition: Inverter system shall perform this diagnosis when "System state"!= [Specific system mode name]. Cycle: [specific processing cycle]
Uncased model	[Software module name] shall judge the execution of following diagnosis depends on following precondition in specified cycle. Diagnosis: Functional restriction because of system power supply low Precondition: Inverter system shall perform this diagnosis when "System state"!= [Specific system mode name]. Cycle: [specific processing cycle]
MNLI model	[Software module name] shall judge the execution of following diagnosis depends on following precondition in specified cycle. Diagnosis: Functional restriction because of system power supply low Precondition: Inverter system shall perform this diagnosis when "System state"!= [Specific system mode name]. Cycle: [specific processing cycle]

the MNLI model mistook some of the "input" components as "output". The models show the model's ability to predict the IO entities in two functional requirements at the same time.

Overall, our 6 representative examples illustrate the ability of our three trained models on recognizing the IO entities in the functional requirement sentences with different writing styles. It illustrates how practical our model could be applied to industrial service for speeding up the analysis. Especially for example 4 and example 5, the writing styles of those sentences are more complicated than the rest with the extra information after the sentence.

5.5 Discussion and Impacts of Our Research

5.5.1 Consideration of Large BERT Model

While we only illustrated the base BERT model in this paper, we also experimented with the large BERT model which has more transformer encoder layers as the official paper demonstrated the better performance of the large BERT model. However, we only had less than 1000 functional requirement records, and the small size of the dataset adversely affects the training stability in a large model including optimization difficulties in

Table 9 Example 5.

Models	Sentence and labels
Ground truth	[Software module name] shall output status of following diagnosis in specified cycle. Diagnosis: Functional restriction because of system power supply low Output: Output the diagnostic status generated by the Fault determination to the [Software module name]. Cycle: [specific processing cycle]
Cased model	[Software module name] shall output status of following diagnosis in specified cycle. Diagnosis: Functional restriction because of system power supply low Output: Output the diagnostic status generated by the Fault determination to the [Software module name]. Cycle: [specific processing cycle]
Uncased model	[Software module name] shall output status of following diagnosis in specified cycle. Diagnosis: Functional restriction because of system power supply low Output: Output the diagnostic status generated by the Fault determination to the [Software module name]. Cycle: [specific processing cycle]
MNLI model	[Software module name] shall output status of following diagnosis in specified cycle. Diagnosis: Functional restriction because of system power supply low Output: Output the diagnostic status generated by the Fault determination to the [Software module name]. Cycle: [specific processing cycle]

early training and vanishing gradients with only a few datasets available. [13], [14] For this reason, we focus on the variants of the pre-trained BERT model in this paper and the stability of a small dataset trained on a large model will be one of the directions for performance improvement in the future.

5.5.2 Improvement of Data Labels

To ensure better performance of the model, we should balance the variety of data labels without superseding specific types of our labels. For example, only relatively few sentences in our fine-tuning datasets have "input" components compared to other types of components due to the limitation of available data and our result shows that this deteriorates the performance of recognizing the corresponding label. In the future, we will spend more resources on ensuring the label variety and further advance the training data generation technique for controlling the number of target types of labels by using different semi-supervised learning techniques such as GAN-BERT. [15]

On the other hand, our current NER trained model can only handle either condition, input, or output relation for each token in the same requirement sentence. There are few exceptional cases for the same component with multiple relations. **Figure 6** illustrates the exceptional case that the same component "B" in the functional requirements could have two different input and output relations to component "A" and component "C" respectively at the same time. Since it means that there are some tokens

Table 10 Example 6.

Models	Sentence and labels
Ground truth	The inverter software shall calculate derating Power depending on DC current. The inverter software shall calculate power using map or calculation
Cased model	The inverter software shall calculate derating Power depending on DC current. The inverter software shall calculate power using map or calculation
Uncased model	The inverter software shall calculate derating Power depending on DC current. The inverter software shall calculate power using map or calculation
MNLI model	The inverter software shall calculate derating Power depending on DC current. The inverter software shall calculate power using map or calculation

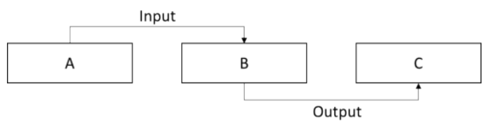

Fig. 6 The example of the same component with multiple relations.

having multiple relations simultaneously, we will further exploit the technique of nested NER [18] in the future to handle the IO entities which have the complex relationships with multiple labels for the same token at the same time.

5.5.3 Consideration of Pretraining Procedure Improvement for BERT Model

Although this paper concentrated on the application of the model pre-trained on general language knowledge to inverter control function in embedded systems due to the limitation of available technical documents, we could pre-train the BERT or other similar NLP based model with the corpus from the requirement documents for the inverter control function in embedded systems as long as the resources are available in the future. Pre-training the model with the corpus in a specific target domain enables the model to extract the deeper underlying features of domain knowledge and therefore improve the fine-tuning accuracy.

A similar concept of pretraining model with the data in a specific domain has already been successfully applied in the medical field although their problem is not the issue of recognizing IO entities presented in this paper. [16], [17] In medical NLP application, the base model was retrained with the corpses of medical articles to improve the accuracy of medical-related tasks such as finding out the symptom and corresponding disease.

5.5.4 The Model for Other Relevant Functional Requirements

Although the paper only spotlighted the application in the functional requirements of the embedded systems implemented with inverter control function, our experiment result implies the

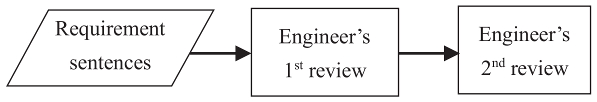

Fig. 7 the original reviewing process without AI model.

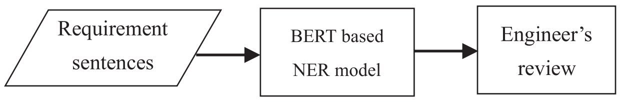

Fig. 8 New reviewing process with AI model involvement.

Table 11 The judgement about NER model result.

Case	NER model result	Judgement about NER model result
1	Less than 2 entities can be recognized	Suspicion of unclear description
2	Either input or output entity can be recognized along with recognized condition entity	Proficient writing quality by addressing the least number of recognizable entities
3	All entities can be recognized	Proficient writing quality without unrecognizable entities

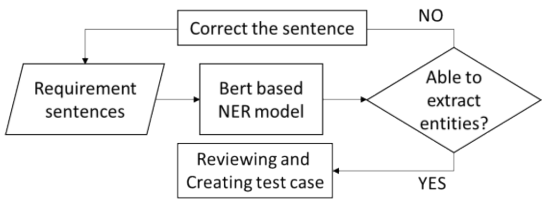

Fig. 9 The improvement of requirement readability.

model pre-trained on general language knowledge is ably applied to a software-related application without critical performance issues and biases. In the future, we will also expand the application in other relevant functional requirements but not only embedded systems with inverter control functions.

5.5.5 The Improvement of Industrial Software Testing Processing

For such kind of IO entities' analysis in requirement sentences, software testing engineers always count on their tacit knowledge as they should comprehend the meanings of the requirement sentences to find out the target entities. Therefore, the junior engineers or the engineers from dissimilar working backgrounds could have different analytical results with the issue of double-standard and we need to squander additional time for ensuring the analysis quality. **Figure 7** illustrates the process without AI model involvement.

Even though our method is yet to be able to provide a completely accurate result when there is no explicitly absolute correct annotation in certain circumstances in which the entities can have more than one identity at the same time, our tool can provide much standardized and consistent analysis for the reference of IO entities recognition based on high-quality annotation data such that the AI model ably captures the most general and acceptable semantic concept without the issue of double-standard. As shown in **Figure 8**, the software testing engineers could skip the first review by relying on our model and therefore shorten much reviewing time by settling the double-standard issue and double-checking the analysis of IO entities from others with highly efficient production.

Additionally, our BERT model can ensure the readability of the requirement sentences by determining whether the target IO entities can be able to be recognized by the AI model. Specifically, we have set the criteria shown in **Table 11** to determine whether the entities are able to be extracted or not from the requirement sentences.

Since the requirement sentences are all prepared in natural language by humans, our BERT-based NER model shown in **Figure 9** can be able to filtrate and recheck the sentences based on the information of unrecognizable IO entities with our criteria such that it could drastically relieve the reviewing process of problematic sentences. This also implies that there should be no more lengthy correction process from requirement sentence writers during reviewing and creating test cases from software testing engineers as they have been resolved in advance with the assistance of BERT based NER model.

6. Related Work

Conventionally, most of the proposed approaches which tackle the accurate analysis of functional requirements are to design the applicable procedure and rules for writing those requirement sentences to efficiently convey the precise information. [1], [2] However, the methods only guided software testing engineers for effective communication but did not automate the whole analysis procedure. In other words, the methods still heavily involved human intervention which is usually prone to grammatical mistakes and relatively expensive compared to the AI automation method proposed in this paper.

In the recent literature of software requirement analysis using AI, most of them are either classifying the whole requirement sentences or simply recognizing the entities without complicated relation of IO entities but they did not address the complicated IO relation issue presented in this paper. [19] utilized the technique of NER with both machine learning and deep learning method to recognize simple software-related entities in software requirement specification (SRS). [20], [21] classifies the application related reviews requirement into given categories with the BERT model for apps performance analysis. The result from [19] further shows that BERT outperformed the rest of machine learning-based approaches.

7. Conclusion

In this paper, our method managed to solve the issue of the IO entities' recognition in the functional requirement sentence dedicated to the inverter control function of embedded systems by fine-tuning the token classification in state-of-the-art BERT model with the small size of exemplary data to conduct the technique of Named entity recognition (NER) and find out the target IO entities from the sequence of tokens.

In our evaluation, we fine-tuned three variants of the BERT models including Cased, Uncased, and MNLI with 829 records and we tested the performance of those three variants of the model with 44 records which consist of 95 functional requirement sentences. From the result. We successfully demonstrated that the IO entities in embedded systems with inverter control function can be interpreted well with the best model achieving around 80% F-measure for all labels. Specifically, the cased and uncased variants of the BERT model have the similarly best performance with more than 80% F-measure for all labels whereas the MNLI variant has the worst performance with less than 80% F-measure for some labels.

This result further provides insight into the fact that the cased and uncased variants of the models pre-trained on general language knowledge are beneficial to the analysis of inverter control function in the embedded system by only preparing small size of fine-tuning datasets for the knowledge of IO entities in the targeted type of embedded systems when the documentation of those embedded systems for pre-training model is not available. Furthermore, the result proves the practicality of applying the NLP model to the targeting industries and therefore improves the industrial productivity by automating the reviewing procedures of embedded system's functional requirements and reducing the man-hour cost of analyzing the functional requirement sentences.

Reference

[1] Mavin, A. and Wilkinson, P.: A. Harwood and M. Novak, Easy Approach to Requirements Syntax (EARS), *2009 17th IEEE International Requirements Engineering Conference*, pp.317–322 (2009).

[2] Verma, R. P. and Beg, M. R.: Representation of Knowledge from Software Requirements Expressed in Natural Language, 2013 6th International Conference on Emerging Trends in Engineering and Technology, pp.154–158 (2013).

[3] Nouvel, D. et al.: J. Wiley & Sons, Appendix 5: Named Entities: Current Definitions, Named Entities for Computational Linguistics, pp.153–156 (2016).

[4] Li, J., Sun, A., Han, J. and Li, C.: A Survey on Deep Learning for Named Entity Recognition, *in IEEE Transactions on Knowledge and Data Engineering*, Vol.34, No.1, pp.50–70 (2022).

[5] Sutton, C. and McCallum, A.: An Introduction to Conditional Random Fields, now (2012).

[6] J., Devlin, et al. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. ArXiv: 1810.04805 [Cs], (2019).

[7] Li, J., Sun, A., Han, J. and Li, C.: A Survey on Deep Learning for Named Entity Recognition, *in IEEE Transactions on Knowledge and Data Engineering*, Vol.34, No.1, pp.50–70 (2022).

[8] Sutton, C. and McCallum, A.: Composition of conditional random fields for transfer learning, In Proc. of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT '05). Association for Computational Linguistics, USA, pp.748–754 (2005).

[9] Gonz'alez-Carvajal, S. & Garrido-Merch'an, E. C.: Comparing BERT against traditional machine learning text classification. ArXiv, abs/2005.13012 (2020).

[10] A., Vaswani, et al.: Attention Is All You Need, Advances in Neural Information Processing Systems, Vol.30, Curran Associates, Inc., Neural Information Processing Systems (2017).

[11] A., Williams, et al.: A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference, Proc. of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol.1 (Long Papers), pp.1112–22 (2018).

[12] Wolf, T., Debut, L., Sanh, V., C., C., Julien, A., Delangue, P., Moi, T., Cistac, R., Rault, M., Louf, J., Funtowicz, S., Davison, P., Shleifer, Platen, von, Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q. and Rush, A. M.: "HuggingFace's Transformers: State-of-the-art Natural Language Processing", arXiv: 1910.03771, (2019)

[13] M., Mosbach, et al.: On the Stability of Fine-Tuning BERT: Misconceptions, Explanations and Strong Baselines, openreview.net, (2020).

[14] T., Zhang, et al.: Revisiting Few-Sample BERT Fine-Tuning, openreview.net (2020).

[15] Croce et al.: GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples, ACL2020 (2020).

[16] T. Goino and T. Hamagami: Named Entity Recognition from Medical Documents by Fine-Tuning BERT (2021).

[17] Rasmy, L., Xiang, Y., Xie and Z. et al.: Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction. npj Digit. Med. 4, 86 (2021).

[18] Li et al.: A Unified MRC Framework for Named Entity Recognition, ACL 2020 (2020).

[19] Malik, G., Cevik, M., Khedr, Y., Parikh, D. and Başar, A.: Named Entity Recognition on Software Requirements Specification Documents, Proc. of the Canadian Conference on Artificial Intelligence. (2021).

[20] Yang, J., Dou, Y., Xu, X., Ma, Y. and Tan, Y.: A BERT and Topic Model Based Approach to reviews Requirements Analysis, International Symposium on Computational Intelligence and Design (ISCID), 2021, pp.387–392 (2021).

[21] Hey, T., Keim, J., Koziolek, A. and Tichy, W. F., NoRBERT: Transfer Learning for Requirements Classification, 2020 IEEE 28th International Requirements Engineering Conference (RE), pp.169–179 (2020).

Chow, Man Yiu received his Computer Science (M.Sc.) degree from The Chinese University of Hong Kong, Hong Kong, in 2018 and He joined Hitachi Ltd. as an assistant researcher in the same year. He is currently working on the research of AI-based requirement engineering improvement and obstacle detection for mobility systems in the DX Engineering Research Department at Hitachi Ltd. His research interests include Natural language processing, Sensor fusion, Machine learning, Data science, and Pattern Recognition.