

A Prompt-based Few-shot Learning Approach to Software Conflict Detection

Robert K. Helmecezi
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
rhelmecezi@ryerson.ca

Mucahit Cevik
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
mcevik@ryerson.ca

Savas Yildirim
Data Science Lab
Ryerson University
Toronto, Ontario, Canada
savas.yildirim@ryerson.ca

ABSTRACT

A software requirement specification (SRS) document is an essential part of the software development life cycle which outlines the requirements that a software program in development must satisfy. This document is often specified by a diverse group of stakeholders and is subject to continual change, making the process of maintaining the document and detecting conflicts between requirements an essential task in software development. Notably, projects that do not address conflicts in the SRS document early on face considerable problems later in the development life cycle. These problems incur substantial costs in terms of time and money, and these costs often become insurmountable barriers that ultimately result in the termination of a software project altogether. As a result, early detection of SRS conflicts is critical to project sustainability. The conflict detection task is approached in numerous ways, many of which require a significant amount of manual intervention from developers, or require access to a large amount of labeled, task-specific training data. In this work, we propose using a prompt-based learning approach to perform few-shot learning for conflict detection. We compare our results to supervised learning approaches that use pretrained language models, such as BERT and its variants. Our results show that prompting with just 32 labeled examples can achieve a similar level of performance in many key metrics to that of supervised learning on training sets that are magnitudes larger in size. In contrast to many other conflict detection approaches, we make no assumptions about the type of underlying requirements, allowing us to analyze pairings of both functional and non-functional requirements. This allows us to omit the potentially expensive task of filtering out non-functional requirements from our dataset.

KEYWORDS

Prompting, Prompt-based learning, PET, few-shot learning, software requirement specification (SRS), conflict detection, transformer models

ACM Reference Format:

Robert K. Helmecezi, Mucahit Cevik, and Savas Yildirim. 2022. A Prompt-based Few-shot Learning Approach to Software Conflict Detection. In *Proceedings of 32nd Annual International Conference on Computer Science and Software Engineering (CASCON'22)*. ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

A software requirement specification (SRS) document outlines the desired behaviour of a new software program. These requirements often specify both functional and non-functional requirements. A coherent SRS is essential to the software development process, as conflicting requirements which go undetected can result in a considerable amount of lost time, particularly if the development process is late into the software development cycle [1, 14]. While some entities can absorb the cost associated with recovering from this time loss, issues with SRS documents can often result in the termination of a software development project altogether [1]. As a result, ensuring that the requirement specifications do not conflict with each other is an essential task in software development. This task is made especially difficult for large software projects where numerous different parties can contribute to the specifications, and where requirements are continually subject to change [7, 26].

In this work, we consider a conflict detection task described as follows. Given an SRS document containing N requirements, the number of pairings of requirements is $O(N^2)$. The task is to assign one of the following labels to each pairing: *Conflict*, indicating that the pairing contains two conflicting requirements; *Duplicate*, indicating that the pairing contains two requirements that are equivalent; and *Neutral*, indicating that the pairing contains two mutually independent requirements. Detecting duplicates is important in part because, if one requirement is to change, it must do so while still satisfying the requirements specified by its duplicates [7]. For particularly large software development projects with many requirements, it is infeasible to label each pairing manually. Instead, we propose to use few-shot learning, specifically through pattern-exploiting training (PET), to label each pairing [20]. PET is a prompt-based learning method which leverages access to a large set of unlabeled pairings, \mathcal{D} , and a small set of labeled pairings, \mathcal{T} . In the conflict detection domain where it is feasible to manually label a few requirement pairings, many of the remaining pairings to be labeled can be used as \mathcal{D} .

PET is built upon pretrained language models (PLMs), such as BERT and its variants, but it is able to classify sequences with high accuracy while using far fewer labeled examples by rephrasing inputs as cloze-style questions. This rephrasing provides context to

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CASCON'22, Nov 15–17 2022, Toronto, Canada
© 2022 Copyright held by the owner/author(s).

the task that is solved, in our case providing the PLM with knowledge of the conflict detection task.

The objective of this research is to establish PET as a viable candidate for few-shot learning for the conflict detection tasks. The contributions of our study can be summarized as follows:

- We extend Malik et al. [12]’s work on conflict detection using transformer models to a few-shot setting. In particular, we provide a direct comparison of BERT and its variants on an array of training sets of size 32 up to 2,048. Our results detail the few shot performance of each transformer model and also quantify the impact of gradually increasing training set size on model performance.
- We investigate the impact of reformulating input examples as cloze-style phrases on few-shot performance. Specifically, we consider reformulations adopted from Schick and Schütze [20, 21] and introduce several of our own patterns. This investigation serves two purposes: firstly, it provides insight into how well generic reformulations transfer into the conflict detection task; and secondly, it provides a comparison of generic reformulations with ones that are targeted specifically towards conflict detection.
- We demonstrate the viability of few-shot learning for conflict detection. Specifically, we show that, given access to a small set of labeled data, prompt-based learning through pattern-exploiting training can produce models which are comparable in terms of well-known performance metrics to supervised learners trained on substantially larger training sets.

The rest of the paper is organized as follows. Section 2 provides a review of the relevant studies in conflict detection and machine learning techniques for SRS text classification. Section 3 provides an overview of our conflict detection dataset, transformer models, and the PET algorithm. Section 4 summarizes the few-shot performance of the considered PLMs and demonstrates the few-shot abilities of PET for the conflict detection tasks. Section 5 summarizes our findings and proposes future research directions as extensions to our research.

2 LITERATURE REVIEW

SRS documents form the premise for a considerable amount of research due to their importance in the software development life cycle. Research into SRS documents includes requirements classification [5, 17], ambiguity detection [14, 23], and fault detection [2, 22]. Many studies also focus on introducing various natural language processing (NLP) techniques into the software requirements domain [5, 6, 9, 13, 23].

Kici et al. [9] investigate text classification techniques for SRS documents using transfer learning through transformer models. In particular, they investigate the performance of BERT, DistillBERT, RoBERTa, ALBERT, and XLNet on several different requirements datasets, some of which contain both functional and non-functional requirements, and several of which contain less than 1,000 total samples. In our analysis, we extend Kici et al. [9]’s comparative analysis by investigating the impact of training set size on the performance of transformer models. While Kici et al.

[9] achieve strong performance across several of their datasets, as their models are specifically fine-tuned on each classification task, their investigation assumes that a substantially large number of requirements for a particular project are already labeled. Our application of the PET algorithm offers a means for training a model using considerably fewer labeled instances than an equally-performant supervised learner.

Conflict detection in SRS documents is a popular research topic [1, 7, 18, 24]. Yang et al. [24] develop RECOMA (REquirements CONflicts MAnagement tool) for both identifying and managing the conflict detection process. Notably, their research attempts to provide a means for conflict detection while acknowledging that formal techniques for developing SRS documents are often infeasible due to the diverse set of stakeholders, many of whom cannot be expected to understand the language that formal methods require [7, 24]. RECOMA relies on a data preprocessing step by proposing a rigid structure for outlining requirements by splitting requirements into an object, a verb, and a resource. This preprocessing is often delegated to a developer, who must then translate requirements to this structure before conflicts can be detected in RECOMA. Additionally, the RECOMA approach only extends to functional requirements.

Guo et al. [7] propose FSARC (finer semantic analysis-based requirements detector), an automated approach to conflict detection for functional requirements front-lined by semantic analysis of documents. Their approach requires first converting requirements to an eight-tuple that forms a harmonized semantic meta-model [7]. Each semantic element in a requirement is identified using a separate algorithm. The identification of semantic elements relies on heuristics and requires the manual labeling of a small number of requirements, where labels can often be ambiguous resulting in discrepancies between multiple human labelers [7]. Guo et al. [7] present promising results for FSARC, indicating that in spite of potential discrepancies between human labelers, the algorithm can still perform well in many circumstances. However, as this algorithm relies on converting requirements to an eight-tuple, in cases where identifying the tuple components is impossible or considerably ambiguous, the performance of the labeling algorithm may suffer considerably.

Malik et al. [12] propose transformer models for conflict detection based on requirement pairings, and show that this approach can achieve significant performance especially when a large number of labeled pairings are available. In this paper, we adapt Malik et al. [12]’s approach to the conflict detection task in a few-shot setting.

To the best of our knowledge, no prompt-based method has been used in SRS conflict detection and, further still, few-shot learning has also not been applied to this task. Prompt-based learning is a relatively new paradigm in the NLP field. The GPT [15] and T5 [16] models are the strongest early examples of prompt-based learning. The GPT-3 [3] model achieves remarkable few-shot performance based on in-context learning by leveraging a natural-language prompt and a few task demonstrations. T5 shows that any NLP problem can be cast as text-to-text, which has been a major breakthrough in this field. Likewise, the autoencoder models reformulate downstream tasks with a masked language model

(MLM) objective. Reformulating is done by adding task-specific tokens to the input sequence for conditioning, which gives us the ability to solve many problems by simply manipulating the input. With prompting, we can even train a model with no access to labeled data as we can directly rely on the objective function (i.e., the MLM objective) [20].

Few-shot learning offers an advantage in conflict detection because it allows using task-specific labelings, particularly labelings for a specific SRS project. Specifically, the prompt-based learning approach that is explored in this study, PET, is able to predict conflicts with high precision using only a few labeled examples, allowing those involved in interpreting and implementing the requirements to spend considerably less time labeling the instances. Additionally, while PET is shown to be useful for few-shot learning, it offers a considerable performance boost even for larger training sets, particularly for difficult-to-label classes. This is despite the fact that the PLMs that are employed by PET are not trained on domain-specific data. Finally, we note that unlike some of the previously mentioned studies, PET does not assume the category of underlying requirements and should be able to label any pairing, including both functional and non-functional requirements. This introduces a particularly powerful advantage over several of the aforementioned studies as it ensures that requirements need not be filtered out based on their type before searching for conflicts.

3 METHODS

In this section we provide a brief overview of the conflict detection dataset, describe the PLMs used in our experiments, and discuss pattern-exploiting training (PET).

3.1 Conflict detection dataset

We employ a proprietary conflict detection dataset obtained from Malik et al. [12]'s work. Given a requirement pairing $x = (R_1, R_2)$ from this dataset, the available labels are *Conflict*, *Duplicate*, and *Neutral*, indicating that the requirements are in disagreement; agreement; or are mutually independent of one another. Table 1 outlines an example for each available label and it also helps demonstrate the difficulties associated with this task. In particular, while we expect it to be relatively easy to detect neutral pairings, the vocabulary used between conflicting and duplicated requirements tends to be very similar.

PET is able to perform few-shot learning in part through the use of a large set of unlabeled requirement pairings, \mathcal{D} . To simulate this unlabeled set, we took 5,000 labeled requirement pairings and discarded their labels. The conflict dataset that we consider is relatively small, containing about 10,000 requirement pairings which are then split between the training, unlabeled, and test sets. In comparison, Schick and Schütze [20] typically considered problems where \mathcal{D} alone had at least 20,000 examples. To split our data, we performed stratified sampling for each of the training, unlabeled, and test sets. The distribution of class labels for each set is outlined in Table 2.

3.2 Transformer models

In our analysis, we deploy several PLM checkpoints, specifically uncased BERT, uncased BERT-large, RoBERTa, RoBERTa-large,

ALBERT-v2, and ALBERT-xxlarge-v2. Each PLM is trained on a large corpus of unlabeled data, and is then fine-tuned on a downstream task [4]. We first evaluate the performance of sequence classification using these PLMs to determine the best performer on our dataset. We then choose the best performing PLM for our prompt-based learning task.

- **BERT (Bidirectional Encoder Representations from Transformers)** models are trained on case insensitive data. These are the base version (BERT-uncased) which has a total of 110M parameters, and the large version (BERT-large-uncased) which has 340M parameters [4]. BERT employs bidirectional self-attention, allowing the model to attend to both previous and future tokens in the self-attention layer [4]. This bidirectional training is made possible through the use of a masked language model [4]. Specifically, in the pretraining objective, tokens are randomly masked and the model must predict the appropriate token based on its context.
- **RoBERTa (Robustly optimized BERT approach)** is a modified version of BERT which implements the changes outlined by Liu et al. [11]. These modifications include increasing the batch size, training for more steps, using a larger pretrain dataset, and employing a more generalized text encoding. In the MLM objective used for BERT pretraining, masking is done as a pre-processing step, which results in the same masked sequences being passed to the model [11]. In contrast, RoBERTa applies masking in a dynamic fashion, generating the masking pattern for each sequence as it is passed to the model [11]. Following from the BERT models, RoBERTa-base corresponds to BERT-base and RoBERTa-large corresponds to BERT-large. The modified text encoding adds parameters to RoBERTa, with RoBERTa-base using 125M parameters and RoBERTa-large using 360M parameters.
- **ALBERT (A Lite BERT)** is another BERT variant, this time designed to improve training times and lower memory usage. ALBERT introduces two major techniques to allow for considerably improved scalability. Notably, ALBERT-base-v2 has just 12M parameters compared to the 110M in BERT-base, and ALBERT-xxlarge-v2 has 235M parameters, less than the 340M in BERT-large [10]. The original ALBERT proposed in Lan et al. [10] is later improved by adjusting several training parameters¹. We employ this updated version, denoted by “v2”, in our work.

In our supervised learning experiments, we employ the sequence pair classification ability of each of the aforementioned language models to label our data.

3.3 Pattern-exploiting training

PET is a prompt-based learning approach introduced by Schick and Schütze [20] for few-shot learning tasks. The contribution of PET to the labeling process is to provide context to the task being solved. Few-shot learning is a method for classifying instances with access to only a small number of labeled examples. This is in contrast to

¹<https://github.com/google-research/albert>

Table 1: Conflict detection examples.

Specification 1	Specification 2	Label
The UAV shall instantaneously transmit information to the Pilot regarding mission-impacting failures.	The Hummingbird shall send the Pilot real-time information about malfunctions that impact the mission.	Duplicate
The UAV shall only accept commands from an authenticated Pilot.	The UAV shall accept commands from any Pilot controller.	Conflict
The UAV flight range shall be at least 20 miles from origin.	The UAV shall be able to transmit video feed to the Pilot and up to 4 separate UAV Viewer devices at once.	Neutral

Table 2: Conflict detection class distribution.

	Conflict	Duplicate	Neutral
$ \mathcal{T} = 32$	17	5	10
$ \mathcal{T} = 64$	33	10	21
$ \mathcal{T} = 128$	67	20	41
$ \mathcal{T} = 256$	134	40	82
$ \mathcal{T} = 512$	267	81	164
$ \mathcal{T} = 1,024$	535	161	328
$ \mathcal{T} = 2,048$	1,070	323	655
\mathcal{D} (unlabeled [†])	2,613	787	1,600
Test	1,045	315	640

[†] Labels are removed for training.

standard supervised learning techniques, which assume access to a large number of labeled training instances. Consider a simplified version of the conflict detection task, where our goal is to label a sentence pair $x = (R_1, R_2)$, as either *Duplicate* or *Not Duplicate*. In a few-shot setting, this is a difficult problem to solve for a standard sequence classifier as it has no context about the task at hand, simply seeing the two sequences. In PET, we provide context by mapping the two requirements to a cloze-style phrase (i.e., we introduce a “prompt”). In this context, we might ask the question:

Does “ R_1 ” imply “ R_2 ”? ____.

where the allowed answers for the masked token “____” in this context are *Yes* for *Duplicate*, and *No* for *Not Duplicate*. In this case, we can use the PLMs with an MLM objective to predict the correct labels.

Figure 1 illustrates an application of the aforementioned context-enriching scenario. In this figure, the objective is to classify the pairing $x = (The\ car\ was\ green.,\ The\ car\ was\ red.)$ as either *Duplicate* or *Not Duplicate*. Notably, these two statements do not immediately fit into the pattern structure: we must perform some preprocessing to remove the terminating punctuation and the initial capital letter. Once this is done, the pattern with the substituted argument x is passed to the MLM. The MLM returns the probability that the masked token “____” should be *No* and the probability

that it should be *Yes*, where *No* indicates that the two statements should be marked *Not Duplicate*, and *Yes* indicates that they should be marked *Duplicate*. These probabilities can then immediately be converted to the label probabilities, and as *No* has the higher probability here, this means the label for this input example is predicted to be *Not Duplicate*.

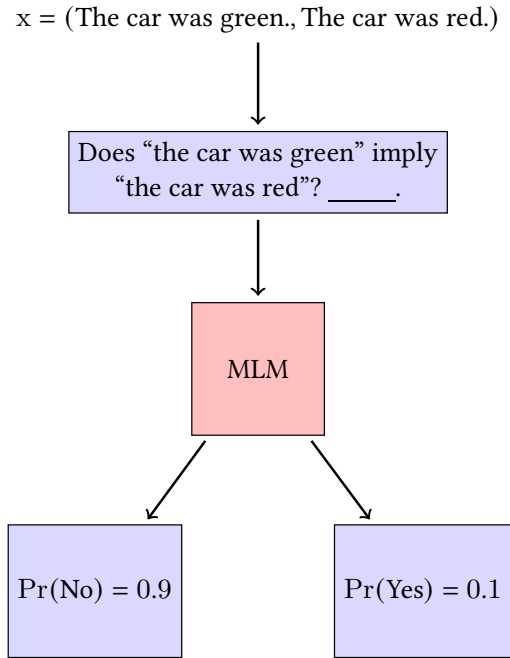


Figure 1: Example application of PET on an input example.

Formalizing this example, we say that a *pattern* P is a mapping of an input x to a cloze-style phrase that contains a masked token [20], which, in the case of conflict detection, can be taken as $x = (R_1, R_2)$. Additionally, we define a verbalizer v as a bijective function which maps each label $\ell \in \mathcal{L}$ to some token or sequence of tokens in the vocabulary of the PLM, \mathcal{V}_{LM} [20]. We can then employ a MLM to predict the most likely substitute for the masked token, and apply v^{-1} to recover the true label. We define a pattern-verbalizer pair (PVP) as $p = (P, v)$ (i.e., it is a tuple of a pattern and a verbalizer).

In theory, PET is able to improve the performance of a learner by adding context to the target problem. In practice however, this is made difficult by several factors. Firstly, in a few-shot setting there is no available data to fine-tune PVPs, allowing for some PVPs to perform worse than sequence classification alone. Secondly, every PVP requires training a separate model, increasing the time required for a PET ensemble to predict the label for a given example x . In the presence of a relatively large set of unlabeled data instances, these issues can be addressed in part through the use of knowledge distillation as outlined by Hinton et al. [8].

Given a set of unlabeled data instances \mathcal{D} and an ensemble of PET models \mathcal{M} , we can use each $m \in \mathcal{M}$ to generate soft labels, or scores, for each $d \in \mathcal{D}$ [20]. These scores are then combined using a weighted sum where the weights are the accuracy of each PVP before training [20]. By using the weight before training, those models which more naturally describe the task even without access to labeled data are assumed to employ superior PVPs. Given the final soft labels, a sequence classifier is trained on the training set together with the softly labeled set \mathcal{D} .

When a MLM m predicts a substitute for the masked token, it assigns a score to each of the tokens in its vocabulary. For our purposes, the important scores are the ones associated with the given verbalizations. In this case, the score assigned to the label ℓ by model m given the pattern $P(x)$ is $s(\ell|x, m)$, and it is simply the score that the MLM assigns to $v(\ell)$. After we train an ensemble of models \mathcal{M} , we can combine the scores for each label $\ell \in \mathcal{L}$ using Equation 1 [20]:

$$s(\ell|x, \mathcal{M}) = \frac{\sum_{m \in \mathcal{M}} w(m) s(\ell|x, m)}{\sum_{m \in \mathcal{M}} w(m)} \quad (1)$$

where $w(m)$ denotes the weight assigned to model m . Finally, we convert label scores to probabilities using softmax.

Our observations, consistent with those in Schick and Schütze [20], show that the performance of a PLM trained on a PVP can vary considerably between runs. As a result, for each PVP we train three separate models. When evaluating PVPs alone we report the average performance of these runs. In our final PET ensemble, we use all three iterations for each PVP selected. This means that for a PET ensemble of three PVPs, there are nine total models contributing to the predictions.

3.4 Patterns

In our investigation, we consider six different patterns: two as proposed by Schick and Schütze [20, 21] (marked with an asterisk in the table), and four of our own. We list all patterns in Table 3. We mark our masked token(s) with a single “_____”. Notably, when a verbalizer consists of more than a single token, we must use multiple consecutive masked tokens as described by Schick and Schütze [21]. In our experiments, all verbalizations require the use of two masked tokens. Additionally, we separate text segments with vertical bars (||). This separation is handled differently by each PLM [20].

Observe that both of the patterns adopted from Schick and Schütze [20, 21] do not include a trailing punctuation mark on R_2 .

Instead, these patterns use the punctuation already available in R_2 . In contrast, all of our patterns strip the trailing punctuation from both R_1 and R_2 . Additionally, we quote the statements that are input into the pattern, ensuring a natural transition between the input and the rest of the pattern.

The patterns provided by Schick and Schütze [20, 21] are basic, contributing only punctuation and a separator. In contrast, we develop patterns which attempt to facilitate the input x as a natural extension of the text. Patterns 5 and 6 additionally attempt to embed the commutative property of the requirements R_1 and R_2 : the label is unaffected when the two sequences are swapped. As PLMs are trained on a finite sequence length, these “commutative patterns” are not always possible. However, for the conflict detection dataset with a sequence length of 256, we found that we never exceeded this maximum sequence length.

3.5 Verbalizers

As with developing patterns, the selection of verbalizers can have a considerable impact on the performance. Without access to a development set, it is difficult to evaluate how well a verbalizer will perform. Schick et al. [19] discuss a method of automatically choosing verbalizers. This may be especially useful for high cardinality tasks where it is difficult to choose any verbalization other than the identity, which maps a label to itself. Notably, however, their results showed that hand picked verbalizers tend to outperform the automatically generated ones. As a result, we develop our own patterns and verbalizers for our classification tasks. They noted that PLMs have a bias towards frequent words, making verbalizations of frequent words superior to verbalizations with rare words [19]. Accordingly, we attempt to use common words in our verbalization.

Table 4 lists the verbalizers that we used for this task. We consider one verbalizer provided by Schick and Schütze [20], marked with an asterisk, and introduce one of our own. Unlike with pattern design, the choice of verbalizer can have a considerable impact on runtime. The time taken to train a model is linear with respect to the number of tokens used in the verbalization. Here we consider verbalizers with a maximum of two masked tokens.

4 RESULTS

In our analysis, we consider labeled training sets sized in powers of two from $2^5 = 32$ up to $2^{11} = 2,048$. We first investigate the performance of six PLMs: BERT and five of its variants. Then, we choose the best performing PLM to serve as the underlying MLM for our PET patterns. We investigate the performance of several PVPs by training three MLMs for each PVP and reporting the average performance on the test set. Finally, we train a PET ensemble consisting of three PVPs and compare its performance to that of the best supervised learner.

4.1 Hyperparameter selection

Our selection of hyperparameters are largely derived from Schick and Schütze [20, 21]. We note that in a few-shot learning setting it is unlikely that a development set for fine-tuning hyperparameters will be available. As a result, and in consensus with Schick and Schütze [20]’s suggestion, we choose hyperparameters

Table 3: PET patterns for conflict detection.

ID	Pattern
$P_1^*(x)$	“ R_1 ”? _____, “ R_2 ”
$P_2^*(x)$	R_1 ? _____, R_2
$P_3(x)$	Given “ R_1 ”, we can conclude that “ R_2 ” is _____.
$P_4(x)$	“ R_1 ” means “ R_2 ”. _____.
$P_5(x)$	“ R_1 ” implies “ R_2 ” and “ R_2 ” implies “ R_1 ”. The previous sentence is correct: _____.
$P_6(x)$	“ R_1 ” is equivalent to “ R_2 ”. Similarly, “ R_2 ” is equivalent to “ R_1 ”. The previous statements are _____.

Table 4: Verbalizers.

	$v(\text{Conflict})$	$v(\text{Duplicate})$	$v(\text{Neutral})$
v_1^*	No	Yes	Maybe
v_2	False	True	Neither

from previous work, adopting the majority of our hyperparameters from Schick and Schütze [20, 21], with one adjustment. As per the findings in Zhang et al. [25]’s study, which note that training for more batches tends to improve performance, we choose to increase the number of training steps from 250 to 1,000 for all learners except for the final sequence classifier trained on the PET ensemble. As this classifier is trained on the soft-labeled dataset \mathcal{D} which has 5,000 examples, we use 5,000 training steps, as in [20]. Note that, as in Schick and Schütze [20] the number of unlabeled examples was more than 20,000 for each task, training for 5,000 steps on our unlabeled set of size 5,000 represents an effective increase in the number of training steps, which is consistent with our adherence to the observations in Zhang et al. [25]. We summarize the hyperparameters used, borrowing the programmatic variable names from the PET repository², in Table 5.

Table 5: Hyperparameters for PET.

Parameter	Value
max_steps	1,000 [†]
gradient_accumulation_steps	4
learning_rate	1e-5
adam_epsilon	1e-8
warmup_steps	0
max_grad_norm	1.0
max_seq_length	256
temperature	2

[†] For training the PET ensemble on $\mathcal{T} \cup \mathcal{D}$, we use 5,000 steps as per [20].

²<https://github.com/timoschick/pet>.

4.2 Supervised learning performance

We first consider the performance of six transformer models—BERT-uncased, BERT-large-uncased, RoBERTa, RoBERTa-large, ALBERT-v2, and ALBERT-xxlarge-v2—on the conflict detection dataset. As PET is an application of these models, we choose the best performing one as the MLM for the PET algorithm. Notably, this experiment evaluates the few-shot performance of these transformer models when they are applied to a sequence classification task.

Due to variance in the training process, we repeat training three times and report the average results. Figure 2 displays the performance of each language model on each training set. The results for $|\mathcal{T}| = 2,048$, particularly the relative ordering of model performance, are similar to what we would expect from each model if they were trained on large training sets [4, 10, 11]. However, this ordering is considerably different for small $|\mathcal{T}|$. As a consequence, we cannot assume that the performance of a PLM for few-shot learning will reflect its performance on full-sized training sets, as was done in Schick and Schütze [21]. Additionally, we observe a more than 25% difference between the best and worst performers when using 32 training instances. As PET is built on-top of a selected PLM, this performance impact would likely be considerable when using the PET algorithm as well.

We found that RoBERTa-large was consistently the best performing model, except for $|\mathcal{T}| = 2,048$ where it performs moderately worse than ALBERT-xxlarge-v2. As a result, we choose RoBERTa-large as our baseline supervised learner to which we compare our PET results. Additionally, RoBERTa-large is chosen as the underlying MLM for the PET.

4.3 PVP evaluation

In addition to the patterns and verbalizers introduced by Schick and Schütze [20, 21], we developed our own in an attempt to more naturally describe the conflict detection task. However, Figure 3 shows that the PVPs that we designed were no more competitive than those that we adopted, and in some cases ours performed even worse. We also observe that many of the PVPs fail to match up to traditional supervised learning, particularly for smaller training set sizes. These results are welcome as they suggest that patterns, like other hyperparameters, can often be borrowed from previous work. Specifically, while the PVPs introduced by Schick and

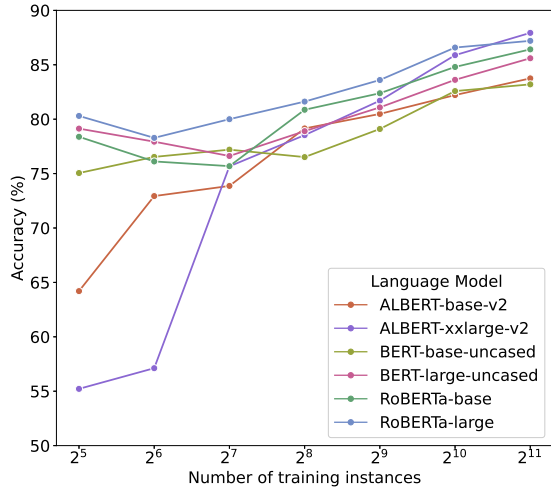


Figure 2: Supervised learning for conflict detection.

Schütze [20, 21] were not targeted to a conflict detection task, they transfer very well into this task with no modifications.

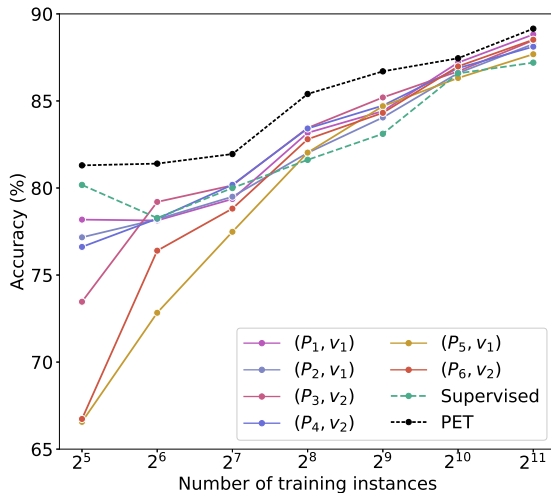


Figure 3: PET for conflict detection.

4.4 Pattern-exploiting training performance

For our PET ensemble, we consider three PVPs: (P_1, v_1) , (P_3, v_2) , and (P_4, v_2) . Due to random variations when training, we follow Schick and Schütze [20] and train three language models for each PVP, resulting in total nine predictors. We then use the ensemble to generate soft labels for 5,000 unlabeled data instances. The soft labels are then used, in conjunction with the labeled training data, to

train a sequence classifier using RoBERTa-large. We observe in Figure 3 that PET outperforms supervised learning on all training set sizes. This is true even for $|\mathcal{T}| = 32$, where none of the predictors in the ensemble surpassed the performance of standard supervised learning.

Table 6 provides a detailed comparison of PET with supervised learning. Recall that RoBERTa-large is both the supervised learner and the underlying MLM for PET. We observe that, for all training set sizes, PET is superior to supervised learning in all of the considered metrics. On average, when using PET instead of supervised learning, we observe an increase of 2.3% in accuracy, 2.9% in macro F_1 score, and 2.2% in weighted F_1 score.

Table 6: PET versus standard supervised learning.

$ \mathcal{T} $	Accuracy		Macro F_1 -score		Weighted F_1 -score	
	Supervised	PET	Supervised	PET	Supervised	PET
32	80.3	81.3	62.8	64.8	75.7	77.1
64	78.3	81.4	67.5	69.0	77.1	79.1
128	80.0	82.0	70.0	71.7	78.9	80.3
256	81.6	85.4	73.4	78.3	80.9	84.6
512	83.6	86.7	75.7	81.0	82.8	86.3
1,024	86.6	87.4	80.4	82.2	86.1	87.2
2,048	87.2	89.2	81.4	84.8	86.7	89.0

Table 7 shows the confusion matrix for $|\mathcal{T}| = 256$, where bolded terms indicate better performance. We observe that PET outperforms or matches the performance of supervised learning in every entry. As we previously noted, differentiating between conflict and duplicate appears to be a more difficult task as each label contains a similar vocabulary. In contrast, neutral statements are often easier to detect as indicated by these results.

Table 7: RoBERTa-large PLM / PET confusion matrix $|\mathcal{T}| = 256$.

True Label	Predicted Label			F_1 -score
	Conflict	Duplicate	Neutral	
Conflict	890 / 941	141 / 97	14 / 7	83.0 / 86.6
Duplicate	199 / 179	112 / 136	4 / 0	39.5 / 49.6
Neutral	10 / 9	0 / 0	630 / 631	97.9 / 98.7

Figure 4 provides a detailed overview of the impact of PET versus supervised learning on a per-label basis. For the *Conflict* label, traditional supervised learning with even 2,048 training samples is unable to match the recall of the PET algorithm with 32 training samples. This is despite the fact that the PET algorithm outperforms supervised learning in precision across all training set

sizes. Additionally, precision for *Conflict* using supervised learning requires 128 training samples to match that of PET with just 32 training instances. In terms of recall, for small training set sizes both PET and supervised learning perform similarly, with supervised learning boasting a slight advantage for small training set sizes before PET widens the gap to nearly 10% as $|\mathcal{T}|$ increases. However, we note that both models have a difficult time recalling pairings labeled *Duplicate* when the training set is small. The precision of PET maintains a 10% lead on supervised learning for all training set sizes when considering the *Duplicate* label. Notably, the performance boost granted by PET for labeling duplicates is substantial even for large $|\mathcal{T}|$, with a boost of around 10% in both precision and recall for the training set of size 2,048. In contrast, the gains for both *Conflict* and *Neutral* tend to subside for large $|\mathcal{T}|$ as they tend to approach their maximum values.

The performance gain in the neutral category is particularly noticeable. We observe that supervised learning requires 1,024 labeled examples in order to match the precision of PET with just 32 examples. Additionally, with access to just 32 labeled examples, PET achieves a precision of 98.6% while recalling 96.6% of examples. This precision margin is nearly perfect and can be conceivably used in a practical setting after training on just 32 labeled examples. This is a valuable result in practice as requirements marked *Duplicate* or *Conflict* both will generally have to be manually reviewed regardless. As a result, using PET with just 32 labeled examples can reliably identify pairings that must be reviewed with incredibly high recall and precision.

5 CONCLUSION AND FUTURE WORK

In this study, we demonstrated the viability of few-shot learning for conflict detection in SRS documents. We have shown that while pretrained language models such as BERT and its variants can be used in a few-shot setting, their performance is substantially worsened when access to data is limited and, furthermore, the relative performance of each model in a few-shot setting is considerably different from a data-rich setting. We also created our own cloze-style reformulations targeted toward conflict detection and compared them with general reformulations adopted from other tasks. Our results showed no considerable difference in performance, suggesting that adopting patterns and verbalizers from other tasks where possible may be sufficient for training a prompt-based learner. Finally, we provided a detailed comparison of PET with the best performing supervised learner and saw that the performance of PET in a few-shot setting was comparable to a supervised learner trained on considerably larger datasets.

We note that our study considers only a single conflict detection dataset, and while our results demonstrate that prompt-based learning is an effective candidate for few-shot learning in the conflict detection domain, future research into a broader array of datasets would be beneficial. Additionally, we note that both the choice of underlying language model for pattern-exploiting training and the selected prompts are both hyperparameters that are difficult to evaluate in a few-shot setting, and which can have a considerable impact on the performance of the final model.

This study offers many natural extensions into future work. Firstly, as noted, this research involves only a single data set. A

more thorough investigation involving additional conflict detection tasks is necessary to establish PET as a viable, generic approach to conflict detection. Secondly, as the use of prompt-based learning for conflict detection is targeted towards few-shot learning, it is unlikely that validation data will be available to gauge the performance of PVPs ahead of time. Accordingly, while our results show that the patterns chosen have a substantial impact on performance, it is left for future research to determine why certain patterns perform better. Thirdly, while this study compares the performance of prompt-based learning with sequence classifiers, it does not compare PET to other few-shot learning approaches. Such a comparison would establish the advantage of using PET in practical settings.

ACKNOWLEDGEMENTS

This research is supported by IBM CAS 1109. The authors would like to thank IBM for providing support and feedback throughout this research. This research was also enabled in part by support provided by the Digital Research Alliance of Canada (alliancecan.ca). We would also like to thank the reviewers for their helpful comments and positive feedback.

REFERENCES

- [1] Maysoon Aldekhail, Azzedine Chikh, and Djamel Ziani. 2016. Software requirements conflict identification: review and recommendations. *International Journal of advanced computer science and applications* 7, 10 (2016).
- [2] Amira A. Alshazly, Ahmed M. Elfatry, and Mohamed S. Abougabal. 2014. Detecting defects in software requirements specification. *Alexandria Engineering Journal* 53, 3 (2014), 513–527. <https://doi.org/10.1016/j.aej.2014.06.001>
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. <https://doi.org/10.18653/v1/N19-1423>
- [5] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software requirements classification using machine learning algorithms. *Entropy* 22, 9 (2020), 1057.
- [6] Hatim Elhassan, Mohammed Abaker, Abdelzahir Abdelmaboud, and Mohammed Burhanur Rehman. 2022. Requirements Engineering: Conflict Detection Automation Using Machine Learning. *INTELLIGENT AUTOMATION AND SOFT COMPUTING* 33, 1 (2022), 259–273.
- [7] Weize Guo, Li Zhang, and Xiaoli Lian. 2021. Automatically detecting the conflicts between software requirements based on finer semantic analysis. *arXiv preprint arXiv:2103.02255* (2021).
- [8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. <https://doi.org/10.48550/ARXIV.1503.02531>
- [9] Derya Kici, Aysun Bozanta, Mucahit Cevik, Devang Parikh, and Ayşe Başar. 2021. Text classification on software requirements specifications using transformer models. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*. 163–172.
- [10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=H1eA7AEtvS>
- [11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
- [12] G. Malik, S. Yildirim, M. Cevik, D. Parikh, and A. Basar. 2022. *Transformer models for conflict identification using requirement pairs*. Technical Report. Toronto Metropolitan University.
- [13] Farhana Nazir, Wasi Haider Butt, Muhammad Waseem Anwar, and Muazzam A. Khan Khattak. 2017. The Applications of Natural Language Processing (NLP) for Software Requirement Engineering - A Systematic Literature Review. In *Information Science and Applications 2017*, Kuinam Kim and Nikolai Joukov (Eds.).

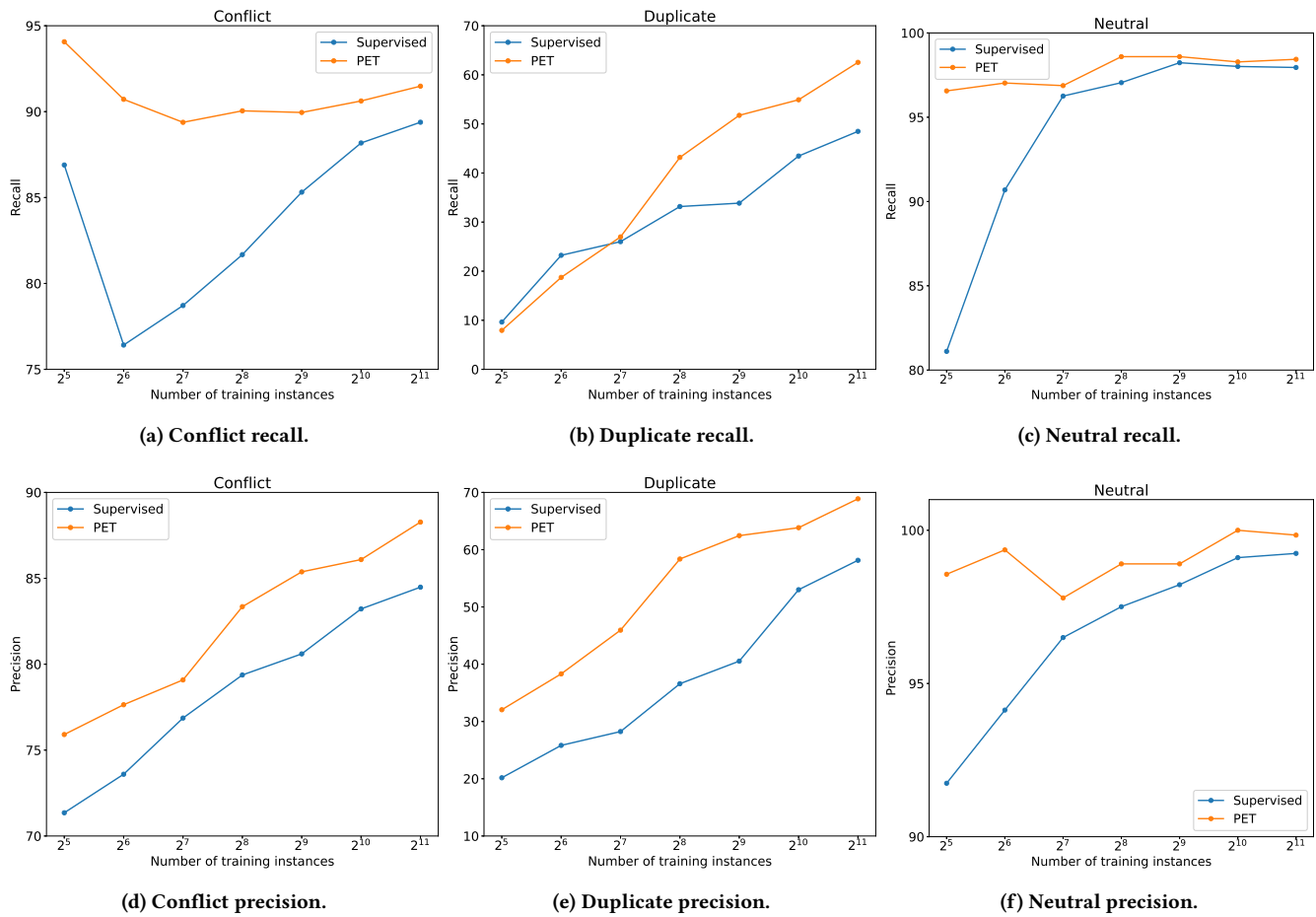


Figure 4: Class-specific recall and precision values.

Springer Singapore, Singapore, 485–493.

[14] Mohd Hafeez Osman and Mohd Firdaus Zaharin. 2018. Ambiguous Software Requirement Specification Detection: An Automated Approach. In *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*. 33–40.

[15] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[16] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21, 140 (2020), 1–67.

[17] Nouf Rahimi, Fathy Eassa, and Lamiaa Elrefaei. 2020. An Ensemble Machine Learning Technique for Functional Requirement Classification. *Symmetry* 12, 10 (2020). <https://doi.org/10.3390/sym12101601>

[18] Alberto Sardinha, Ruzanna Chitchyan, Nathan Weston, Phil Greenwood, and Awais Rashid. 2013. EA-Analyzer: automating conflict detection in a large set of textual aspect-oriented requirements. *Automated Software Engineering* 20, 1 (2013), 111–135.

[19] Timo Schick, Helmut Schmid, and Hinrich Schütze. 2020. Automatically Identifying Words That Can Serve as Labels for Few-Shot Text Classification. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, Barcelona, Spain (Online), 5569–5578. <https://doi.org/10.18653/v1/2020.coling-main.488>

[20] Timo Schick and Hinrich Schütze. 2021. Exploiting Cloze-Questions for Few-Shot Text Classification and Natural Language Inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, Online, 255–269. <https://doi.org/10.18653/v1/2021.eacl-main.20>

[21] Timo Schick and Hinrich Schütze. 2021. It’s Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 2339–2352. <https://doi.org/10.18653/v1/2021.naacl-main.185>

[22] T. Thelin, P. Runeson, and C. Wohlin. 2003. An experimental comparison of usage-based and checklist-based reading. *IEEE transactions on software engineering* 29, 8 (2003), 687–704.

[23] Ashfa Umer and Imran Sarwar Bajwa. 2011. Minimizing ambiguity in natural language software requirements specification. In *2011 Sixth International Conference on Digital Information Management*. IEEE, 102–107.

[24] Hwasil Yang, Minseong Kim, Sooyong Park, and Vijayan Sugumaran. 2005. A process and tool support for managing activity and resource conflicts based on requirements classification. In *International Conference on Application of Natural Language to Information Systems*. Springer, 114–125.

[25] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2021. Revisiting Few-sample {BERT} Fine-tuning. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=c01IH43yUF>

[26] Didar Zowghi and Vincenzo Gervasi. 2002. The Three Cs of requirements: consistency, completeness, and correctness. In *International Workshop on Requirements Engineering: Foundations for Software Quality, Essen, Germany: Essener Informatik Beitage*. 155–164.