ConfigMiner: Identifying the Appropriate Configuration Options for Config-related User Questions by Mining Online Forums

Mohammed Sayagh, Ahmed E. Hassan

Abstract—While the behavior of a software system can be easily changed by modifying the values of a couple of configuration options, finding one out of hundreds or thousands of available options is, unfortunately, a challenging task. Therefore, users often spend a considerable amount of time asking and searching around for the appropriate configuration options in online forums such as StackOverflow. In this paper, we propose ConfigMiner, an approach to automatically identify the appropriate option(s) to config-related user questions by mining already-answered config-related questions in online forums. Our evaluation on 2,062 config-related user questions for seven software systems shows that ConfigMiner can identify the appropriate option(s) for a median of 83% (up to 91%) of user questions within the top-20 recommended options, improving over state-of-the-art approaches by a median of 130%. Besides, ConfigMiner reports the relevant options at a median rank of 4, compared to a median of 16-20.5 as reported by the state-of-the-art approaches.

Index Terms—Configuration, user questions, online forums, stackOverflow.

1 Introduction

Configuration options allow users to change and customize the behavior of a software system without the need for source code modifications. For example, one can enable/disable the execution of *javascript* in Firefox[1] by simply switching on/off the configuration option *"javascript.enabled"*[2].

While configuration options make a software system flexible, finding the appropriate set of options for a given situation is not trivial. That is because highly configurable software systems tend to have a vast number of configuration options. Firefox, as an example, has more than 2,000 configuration options, while Hadoop has 318 available options just for its core component. Understanding the goal and meaning of a configuration option is a challenging task [18].

Users often ask others about appropriate configuration options on online forums, such as StackOverflow, ServerFault, or even dedicated forums. However, one often must wait a long time before getting an answer with the appropriate options.

Finding the appropriate options to a given situation is an essential task to configure a software system correctly and avoid configuration errors. These errors are perceived to be frequent, severe, and hard to debug [31]. Recently, due to an incorrect configuration of permission options in an Amazon web service, data of millions of users were publicly exposed[3]. Furthermore, a misconfiguration required more than an hour to get fixed in a Facebook outage [12].

Prior research efforts mostly focus on debugging configuration errors [2], [18], [24] using source code analysis approaches that often start from an actual error message. However, users might not always have a clear error message or exception trace at hand instead they might face a silent error or a functional misbehavior. Even more, users may not necessarily have a problem to fix, instead they might be simply interested in exploring or learning about how to configure a feature or the configuration of a functionality.

A few prior efforts [10], [25] explored helping users identify appropriate options. Given a config-related user question, Jin et al. [10] and Wen et al. [25] identify options that are textually similar to the user question. That textual similarity is calculated based on the number of relevant words that each option name shares with the user question. Such relevance is quantified using the TF-IDF technique[4]. However, a config-related question does not necessarily have to share words with its appropriate options, and it might even share words with inappropriate option names. We observed that option names are often not clear enough, do not necessarily represent the feature that they are configuring, and maintaining a *"workable naming convention for all configuration options"* is not trivial [18].

Therefore, this paper aims at gaining a deeper understanding of config-related user questions via a qualitative and quantitative analysis of online user questions. Second, this paper proposes an approach to identify the appropriate options for a config-related question, which we evaluate and compare to prior state of the art approaches by Jin et

• *Mohammed Sayagh and Ahmed E. Hassan are with the School of Computing, Queens University, Canada.*
E-mail: msayagh@cs.queensu.ca, ahmed@cs.queensu.ca.

1. https://www.mozilla.org/en-US/firefox/
2. https://www.technipages.com/firefox-enable-disable-javascript
3. https://www.pcworld.com/article/226033/thanks_amazon_for_making_possible_much_of_the_internet.html
4. http://www.tfidf.com/

al. [10] and Wen et al. [25] as well as a basic Google search approach. We summarize our contributions as follow:

- A qualitative analysis to better understand the types of config-related questions that users ask in forums. We found that in addition to questions about errors with an explicit message or exception trace, users commonly (25% of the studied questions) ask about how to solve misbehaviors that do not have any associated error messages (such messages can be used as input for existing configuration debugging approaches, e.g., [6], [17]), and ask about how to configure a functionality in general.
- A quantitative analysis of 2,062 configuration conversations to understand the elapsed time to receive an answer with the appropriate options. We found that one can wait for a median of around 4.75 hours before getting the appropriate answer, which suggests that even if one might receive an accurate answer in an online forum, waiting for an answer is time consuming for the three types of config-related question.
- An approach (called ConfigMiner) that identifies the appropriate options to a config-related question. Given a new question, ConfigMiner identifies the already-answered config-related questions that are similar to that new question, and recommends any configuration options that are associated with these already-answered questions. Our evaluation of ConfigMiner on 2,062 online questions from 7 software systems shows that it can report the appropriate options for a median of 83% of the user questions within the top-20 recommended list of options (outperforming the state-of-the-art approaches; PrefFinder by Jin et al. [10] and CoLUA by Wen et al. [25] by a median of +130%).

The paper is organized as follow. Section 2 presents the background and discusses related work. Section 3 presents our qualitative and quantitative analysis of config-related questions on online forums. Section 4 and Section 5 present ConfigMiner and the baseline approaches respectively. Section 6 reports our empirical evaluation of ConfigMiner and its comparison to the baseline approachees. Section 8 discusses threats to the validity of our observations. Finally, Section 9 concludes the paper.

2 Background and Related Work

One can tailor a software system and adapt it to different situations and contexts using configuration options. Configuration options are a set of key and value pairs available to end-users, where the key is a configuration option name and the value represents a user choice for that option. For example, one can cache DNS lookups in Firefox by changing the value of the *"network.dnscacheexpiration"* option.

Two main lines of research on software configuration exist [18]. While the first line of research focuses on preventing configuration errors, the second line consists of techniques that help debug configuration errors. Few research efforts [10], [25] identified the appropriate configuration options for config-related questions.

2.1 Empirical Study on Configuration Errors and their Impacts

Configuration errors are one of the most commonly reported errors [31]. Configuration options add more complexity to developing and testing a highly configurable software system [11]. As much as 59% of the performance bugs are caused by configuration errors [8]. Moreover, configuration options can have an impact not just on the software system to which they belong but on other layers of a given stack of software systems such as the LAMP stack [16], which in turn leads to severe and time-consuming errors, referred to as cross-stack configuration errors [17].

2.2 Debugging Configuration Errors

Prior efforts on debugging configuration errors use source code analysis approaches that require as a starting point an error message or a previous correct execution trace [2], [18], [24]. For example, Attaryian et al. [3] used dynamic control and data flow analysis to report the culprit options. Dong et al. [6] used backward and forward slicing to find culprit options. Zhang et al. [32], [33] proposed an approach that compares the traces of an incorrect execution against existing traces of correct executions to identify which options lead to execution variances. In our prior work [17], we proposed a modular approach that combines existing source code analysis techniques to find misconfigured options in a stack of application layers. For more details about configuration errors and existing approaches to debug them, we refer to our previous systematic literature review [18], as well as surveys of configuration errors [2], [24] who summarize the existing configuration debugging approaches.

While those approaches focus on debugging configuration errors, they need a starting point that is often an error message or a previous correct execution trace. However, users might not always possess such an execution trace and might not have an error message at hand. A user might be simply facing a functional misbehavior that is not producing any error message or exception traces. One might also wish to simply understand how to customize a particular behavior or what are the appropriate options to a feature. Therefore, this paper studies the different types of config-related user questions and proposes an approach to identify their appropriate options.

2.3 Identifying the Appropriate Configuration Options for Config-related Questions

A config-related user question is a question whose answer contains a set of appropriate configuration options [10], [25]. Figures 1 shows a user config-related questions on StackOverflow. The user had a problem related to the cursor shown in the text input, which required him to change the value of the *"bidi.browser.ui"* option that is suggested by the accepted answer. We consider that the *"bidi.browser.ui"* option is the appropriate option for that user question.

While most prior studies focus on how to debug configuration errors, Jin et al. [10] and Wen et al. [25] proposed approaches for identifying the appropriate options to a config-related question. Both of these prior approaches recommend options whose names are textually similar to the new user

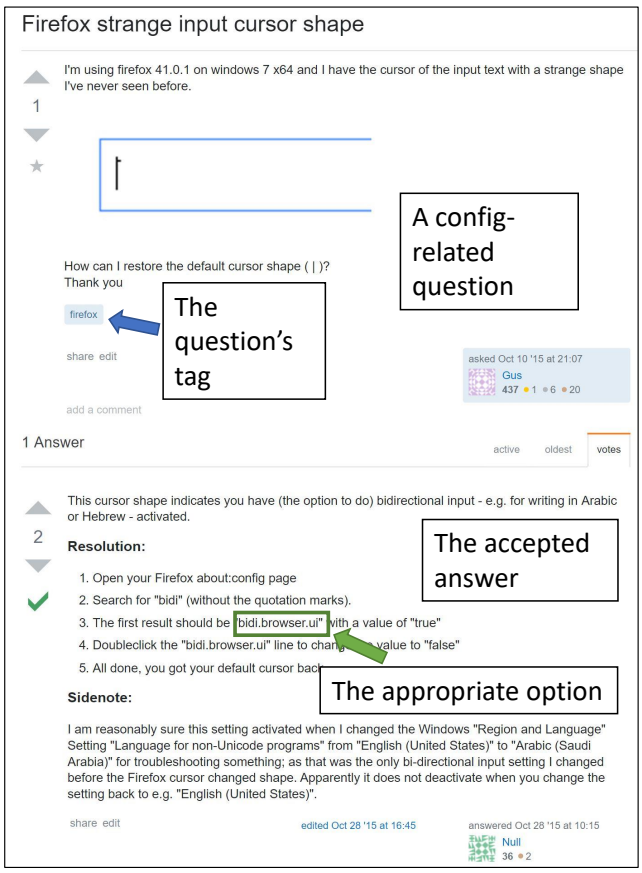

Fig. 1. An example of a StackOverflow question and its appropriate option.

question. While Jin et al. [10] considered a user question as the title of a question asked in an online forum, Wen et al. [25] considered a user question as the title and the body of a bug report. More details about their proposed approach are discussed in Section 5.

We believe that approaches which identify the appropriate options using their constituting words can face some limitations. As discussed earlier, an option might not necessarily share words with the user question, a question can contain words that are common with other unrelated options, and developers may not follow a clear naming convention such that option names expressively represent what they configure. For example, none of the words that constitute the appropriate option *"bidi.browser.ui"* exists in the user question of Figure 1.

Already-answered questions might have previously solved a question that is similar to the new config-related question. Therefore, we propose an approach that learns from already-answered questions to identify the appropriate options for a new config-related question. Section 4 details our approach that we evaluate and compare to PrefFinder and CoLUA, which are the baseline approaches by Jin et al. [10] and Wen et al. [25] in Section 6.

2.4 Duplicated Bug Reports and StackOverflow Questions

Another related line of research focuses on detecting duplicated bug reports or StackOverflow questions. For example, Zhang et al. [34] proposed an approach that predicts if a new StackOverflow question is duplicated using a classification algorithm and different textual similarity algorithms (e.g., doc2vec, and topical similarity). Ahasanuzzaman et al. [1] extended Zhang et al.'s work by considering additional features for the classification algorithm. Fu and Menzies [7] found that a simple method (Word embedding using word2vec and SVM) can quickly achieve similar (and sometimes better) results compared to a more sophisticated approach (Word embedding and convolution neural network, which was proposed by Xu et al. [29]) for identifying the semantic link between two StackOverflow questions. Sun et al. [21] used the IDF part of the TF-IDF technique and a classification model to identify duplicate bug reports. Boisselle et al. [4] used TF-IDF to identify duplicate bug reports cross the bug repositories of related projects (e.g., Ubuntu and Debian).

Our paper is different from this line of work. Our goal is not to propose a novel approach that identifies duplicate or similar questions or bug reports. We instead evaluate if using already-answered questions can help identify the appropriate options for a new config-related question, while any of the previous textual similarity approaches can be plugged to our approach as discussed in Section 4.

3 Understanding Config-Related Questions on Online Forums

This section studies config-related questions on online forums, such that we can identify if there is a need for an automated approach to identify the approrpiate option for a config-related question then design an approach that identifies the appropriate options for a new config-related question. To achieve this goal, we conducted a qualitative and quantitative analysis to understand the types of config-related questions that users ask on online forums, and the elapsed time before these questions receive an answer.

3.1 Data Selection

In this paper, we focus on seven software systems (listed in Table 1) and their respective config-related questions that are asked on StackOverflow, ServerFault, and Firefox forums. We focus on these systems since they are popular systems and have a large number of configuration options. We study questions on StackOverflow and ServerFault given their respective popularity among StackExchange forums[5]. We also study questions on the Firefox Forum as it is a well-structured website that we can easily crawl. We used the following criteria to obtain config-related questions:

> **Criterion 1:** For StackOverflow and ServerFault, we selected questions whose tags refer to one of the studied software systems. For example, the question shown in Figure 1 is tagged with *"firefox"*.
> **Criterion 2:** We selected questions whose **accepted answer** contains a configuration option name, such that we can focus on questions that are addressed by configuration options (config-related questions). For Firefox Forum, we selected the **chosen answer**

5. https://stackexchange.com/sites

TABLE 1
Evaluated Software Systems. SO and SF refers to StackOverflow and ServerFault respectively.

Software System ↓	Description	Number of Options	Number of Discussed Options	Number of questions	Data Source
Cassandra	A NoSQL Database	162	64	246	SO (240) and SF (6)
Firefox	A web browser	2,165	252	789	Firefox Forum (627), SO (160), and SF (2)
Hadoop	The common core of Hadoop that supports various modules such as MapReduce	318	35	175	SO (172) and SF (3)
HDFS	A distributed file system	426	54	143	SO (141) and SF (2)
MapReduce	A framework used to distribute the processing of big data to multiple environment	207	60	152	SO (152) and SF (0)
Spark	A distributed computing framework	217	109	426	SO (425) and SF (1)
Yarn	A framework that schedules distributed jobs	397	41	131	SO (131) and SF (0)
Total		3,892	615	2,062	

which is the equivalent to **accepted answer** in StackExchange forums. For example, *"bidi.browser.ui"* is the appropriate option for the config-related question in Figure 1 as the accepted answer mentions that option. We used the accepted answers since they represent the best solution for a question[6]. Note that we obtained the list of all available option names from the official documentation of each of our studied software systems.

We obtained 2,519 questions that respect the previous criteria from the whole data set of StackOverflow, ServerFault, and the 24,875 last questions in Firefox forum.

In addition to considering an option as appropriate to a question when that option is mentioned in the accepted answer, we also applied two additional criteria on the appropriate options for a question:

Criterion 3: We ignore options that are mentioned inside a long snippet of code because the answerer can mention an option which does not specifically address the question. Thus, we ignore snippets with more than 20 lines of code. 82 questions are excluded due to this criterion. 6 accepted answers mention just a subset of the options in large code snippets.

Criterion 4: We do not consider an option as appropriate when it is mentioned in the question as well as the accepted answer. That is because we observed cases where users have just typos in their configuration files and the accepted answer simply highlights that typo. In other cases, users already know their appropriate options but do not know how to change their values or the location of the configuration files. 357 config-related questions are excluded by this criterion, where 117 questions have some of the answers' options within the questions.

Our final data set contains 2,062 out of 2,519 questions with their respective appropriate option as shown in Table 1. Similar to Xu et al. [30], we observed that users do not discuss all the configuration options of a software system, although the number of discussed options is not negligible (between 35 of 252) as shown in Table 1.

6. https://stackoverflow.com/help/someone-answers

3.1.1 Data Selection for the Qualitative Analysis

Our qualitative study focuses on 298 config-related questions and their accepted answers. Initially, we selected a representative random and weighted (by the number of questions for each case study) sample of 324 from the 2,062 questions (confidence level = 95%; confidence interval = 5%)[7], from which we manually filtered out 26 questions that are not related to configuration, ending up with 298 config-related questions. The accepted answers of those 26 questions do mention an option, but it is not appropriate to the question. That said that our data selection approach has an accuracy of 92 % 5 % (298/324) for automatically identifying config-related questions. Section 3.2 provides and discusses the results of this manual analysis.

3.1.2 Data Selection for the Quantitative Analysis

The goal of our quantitative analysis is to measure the elapsed time to receive the appropriate option(s) to a config-related question with the goal to evaluate whether receiving an answer from an online forum is time-consuming and whether the types of config-related questions (identified in the qualitative study) are as time-consuming as each other and require similar attention. Concretely, we measure the time between the creation date of the question and the creation date of its accepted answer. Our analysis considers the 2,062 questions as well as the 298 manually studied questions.

We also compare if there are any differences between the elapsed time to receive the accepted answer cross the 7 different case studies, StackExchange forums, and types of config-related questions using the Wilcoxon test. It is a nonparametric statistical test that compares whether the distributions of two groups are statistically different. A small p-value (lower than) in this context indicates that the probability that the two distributions are similar is low. Hence, if the p-value is lower than , we conclude that the two distributions are statistically different [19].

7. Confidence level and confidence interval indicate how confident one can be about the statistical results that he obtained from a sample. In a simple way, if we repeat the same experiment on 95% (Confidence level) of the possible samples, we obtain the same statistical results with a margin-error of $\pm 5\,\%$ (Confidence interval) [19].

Title: Page scrolling is not smooth in firefox
Body: *When I use Fierfox I have noticed when I'm using my mouse to scroll the page I'm on the page tends to stick or not move for a moment. I just tried IE and no problem there. I was even on the same page and Firefox still trends to hang up. Anyone can help me out?*
Note that the website is mad with classic asp.

Fig. 2. An example of an error without any explicit message.

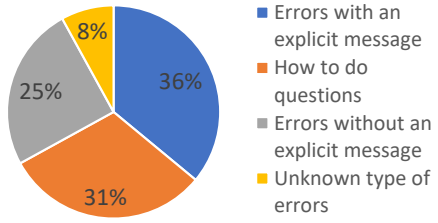

Fig. 3. The distribution of the types of config-related questions.

3.2 Types of Config-related Questions

Users ask different types of config-related questions that are not necessarily addressable by state of the art techniques that use source code analysis techniques. We observe from our qualitative analysis on 298 config-related questions that the most common category (69%) of questions are errors, which we further classified into two subcategories. As shown in Figure 3, the largest (36%) type of errors are those for which the asker have an explicit error messages, warnings, or exception traces. The second largest category (25%) are errors for which the asker did not have any explicit error message, warning, or exception trace. 87% of this last type of config-related questions are related to functional misbehavior and 13% are related to performance degradation. Figure 2 shows a concrete example of a Firefox misbehavior question, which did not have any particular error message. Note that we were not able to classify 23 (8%) errors in these two categories from their textual descriptions.

We found a significant number of config-related questions (31%) that are not problems, but simply inquiries about how to configure a functionality, or enable/disable a feature. Figure 4 shows a concrete example in which the asker inquiries about how to disable the drop-down menu in the address bar of Firefox.

3.3 Elapsed Time for the Accepted Answer for a Config-related Question

Config-related questions receive their accepted answer in 4.75 hours (median) based on our analysis to our 2,062 config-related questions. This median ranges from 2.96 to 17 hours for Firefox and Yarn questions. Besides, we did not observe any statistically significant differences between the elapsed time to get the appropriate option through all the 7 case studies (Wilcoxon test; *p-value* > *0.019;* = *0.01*). However, one might receive a faster answer via Firefox forum compared to StackOverflow (2.67 and 9.17 hours respectively). Note that this last difference is statistically significant (Wilcoxon test; p-value = 4.824e-05; = 0.01).

Prior research efforts studied configuration errors with an explicit message, while the two other types of config-related questions are also as common and as time-consuming as that first type of config-related questions. Our investigation on the 298 manually studied questions shows that all three types of config-related questions require similar elapsed time to receive their accepted answer (Wilcoxon test, *p-value* > *0.5;* = *0.01*). Surprisingly, we did not find any statistically significant difference between the time required to identify the appropriate option for errors with an explicit message and errors without an explicit message (Wilcoxon test, *p-value* = *0.5;* = *0.01*), although one would intuitively expect that an explicit message would help users find the appropriate option faster. We also observe that errors with an explicit message are as time-consuming as *how-to-do* type of questions (Wilcoxon test, *p-value* = *0.6;* = *0.01*). That suggest that even if one might receive an accurate answer in an online forum, it is still time-consuming for the three types of config-related questions.

Title: How to completely disable the drop-down menu in the address bar?
Body: I'd like Mozilla to not show me anything when I use the address bar, other than what I type into it.

Fig. 4. An example of *"how-to-do"* config-related question.

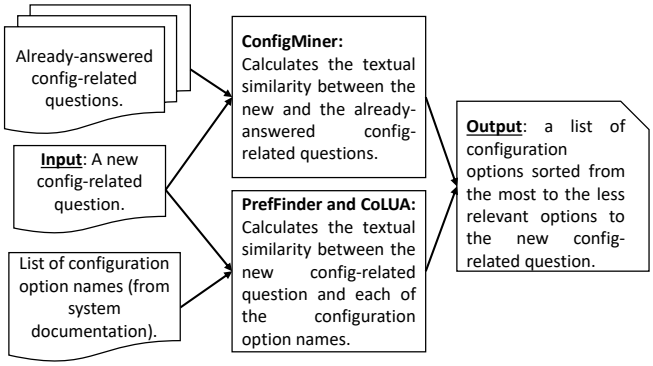

Fig. 5. The differences between ConfigMiner and the baseline approaches (PrefFinder and CoLUA).

There are three types of config-related questions, two (errors with explicit message and how-to-do questions) of which might not be addressed by a source code analysis approach. These two categories are common and require similar amount of time to be correctly answered compared to errors with explicit message.

4 Our Approach for Identifying the Appropriate Options to a Config-related Question

We propose an approach that learns from already-answered config-related questions to identify the appropriate options for a new config-related question. We refer to our approach as ConfigMiner, which we discuss in this section first before presenting our baseline approaches in the next Section.

4.1 ConfigMiner

ConfigMiner follows three main steps to recommend the appropriate options for a config-related question, as shown in Figure 5. ConfigMiner textually compares **a new config-related question (the input)**, which consists of a title and a

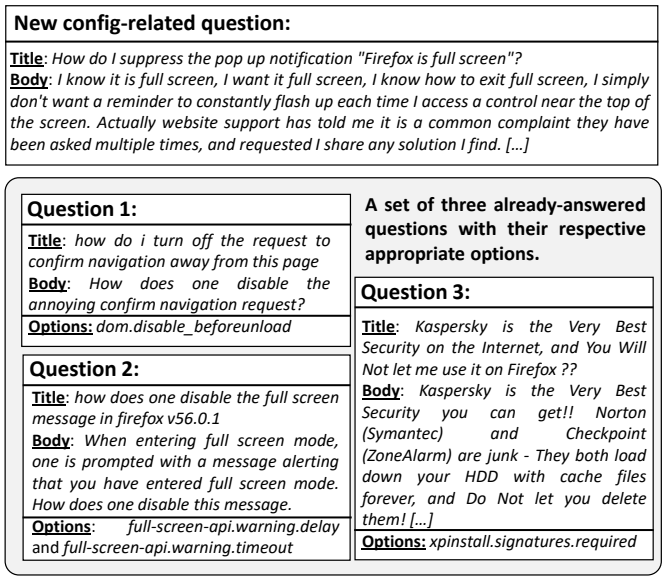

Fig. 6. Example of a new question and a sample of three already-answered questions with their respective appropriate options (solution). In this example, the most similar question to the new question is Question 2. Therefore, we first recommend its appropriate options ("*full-screen-api.warning.delay*" and "*full-screen-api.warning.timeout*").

body, to a set of already-answered config-related questions. It sorts these questions based on their textual similarities to the new question. Finally, **it outputs a set of appropriate option(s)** to the new question. The list of options respects the same order of their respective questions.

Figure 6 shows an example of a new question and a sample of already-answered config-related questions with their respective solutions (i.e., appropriate options). The question and each of the three questions is represented by a title and the body of the question. Furthermore, each of these three questions is associated with its appropriate option(s). For example, the *"dom.disable_beforeunload"* option is appropriate to Question 1.

For the same example and based on a textual similarity between that new config-related question and each of the already-answered config-related questions, ConfigMiner finds that Question 2 is the most similar to the new question. Indeed, they are both about how to disable the full-screen warning message. Therefore, the two options of Question 2 are recommended to address the new question.

A.1. Data Collection: We collected the config-related questions by following the same approach and the four criteria that we discussed in Section 3.

A.2. Textual Similarity: ConfigMiner uses natural language processing (NLP) techniques to calculate the textual similarity between a new config-related question and the already-answered config-related questions (note that ConfigMiner considers both the title and body of a config-related question). While one can calculate textual similarity using different techniques, we opted for using the TF-IDF technique to calculate the textual similarity in this paper for two main reasons. (1) As noted by Fu and Menzies [7] and Liu et al. [14], one should evaluate simple techniques first, since they can report the same (and sometimes better) results compared to more sophisticated techniques. In fact, we evaluated as two additional textual state-of-the-art embedding techniques (the doc2vec [13] and the universal sentence encoder techniques [5]) and TF-IDF outperformed both of these techniques. (2) In addition, using TF-IDF to measure the similarity enables us to perform a more systematic comparison of our approaches performance relative to our baseline approaches (which both use TF-IDF). Hence, we are able to evaluate one variance (using the already-answered questions versus the configuration options names), as shown in Figure 5. Finally, we note that one can plugin any existing or new textual similarity techniques to our approach.

a. Preprocessing Data: Our first preprocessing step consists of removing stop-words such as *"and"*, *"I"*, or *"does"* as well as large source code snippets (at least 20 lines of code).

After removing stop-words, we stemmed all the tokens such that TF-IDF does not consider similar words as different tokens. As a word can be used in different forms, we stemmed all the tokens of our conversations. For example, *"disabled"*, *"disables"*, and *"disabling"* refer to the same word, which is *"disable"*. Therefore, the stemming transforms all the previous four words to a single token that is *"disabl"*.

b. Embedding: TF-IDF first transforms a document to a set of words, each of which has a weight that depends on its frequency in the whole corpus of documents, such that frequent words have less importance compared to particular words to a given conversation. For example, a token like *"browser"* is not appropriate to a specific conversation in Firefox questions as it might appear in a large number of questions, while a token like *"Kaspersky"* can be more specific to a particular Firefox question. Thus, TF-IDF gives a high weight to *"Kaspersky"* and a low weight to *"browser"* using the following equation:

$$idf_{term} = log(\frac{N}{df_{term}}) + 1 \quad (1)$$

where N refers to the number of documents (forum questions) and df_{term} (i.e., *document frequency*) to the number of questions that contain *"term"*.

Furthermore, for a given document TF-IDF gives more weight to a term that is mentioned multiple times and less weight to infrequent terms in a given document *"d"* following this equation:

$$tidf_{term;d} = tf_{term;d} \quad idf_{term} \quad (2)$$

Finally, TF-IDF normalizes the weight of a token based on the length of a document using this equation:

$$weight_{term;d} = \frac{tf\,idf_{term;d}}{\sqrt{\sum tf\,idf^2_{term_i;d}}} \quad (3)$$

Thereby, each document (i.e., question) is embedded into a numerical vector that represents its tokens with their respective weights.

c. Calculating the Similarities: Finally, the similarity score between a config-related question and each of the already-answered questions is the cosine similarity between their two respective vectors, which is defined as:

$$similarity(d1; d2) = \frac{d1\ d2}{jjd1jj\ jjd2jj} \quad (4)$$

5 The Baseline Approaches

We consider PrefFinder [10] and CoLUA [25] as well as a Google based search as baseline approaches.

5.1 PrefFinder and CoLUA

While ConfigMiner compares a question to already-answered config-related questions, **PrefFinder [10] and CoLUA [25] compare the textual similarity between a question and each of the existing configuration option names**, as shown in Figure 5. The baseline approaches do not consider the already-answered questions. They consider that the more an option name is similar to the question, the more appropriate that option is to that question. In other words, a TF-IDF document for the baseline approaches is an option name, while our TF-IDF document is a question or an already-answered question for ConfigMiner.

Apart, the baseline approaches follow the same TF-IDF technique that we described before with some additional data preprocessing and using a different similarity equation, which we detail in the remaining of this subsection.

While PrefFinder considers just the title of questions, CoLUA considers the title as well as the body.

a. Prepocessing Data: While our approach compares the similarity between a question and already-answered questions, the baseline approaches compare the question and each of the existing option names. That said, an option name should be represented by a vector based on the constituting words of that configuration option name.

A typical option name is a set of attached words (1) separated by a delimiter such as dot(.), dash(-), or underscore(_), (2) follows a camel case, or (3) attached without any explicit separator. While a simple split algorithm can easily split the first two patterns, the third one is not straightforward. For example, *"browser.warnOnQuit"* can be easily split to *"browser warn On Quit"* considering the delimiter (dot) and the camel case. However, the token *"beforeunload"* in the configuration option name *"dom.disable_beforeunload"* is difficult to split to *"before unload"*.

To address such cases, Jin et al. [10] adopted the *"backward greedy algorithm"*, which shows accurate splitting results [9]. We refer to the work of Jin et al. [9] for more details about the algorithm and its evaluation.

Similarly to our approach, we also performed stop-words removal, large snippets of code removal, and stemming of option tokens as well for the question.

b. Embedding: The baseline approaches use the same embedding approach that ConfigMiner uses, however on option names and their tokens (that are obtained from the prepossessing step) and not on the config-related question as our approach does.

c. Calculating the Similarities: Jin et al. [10] and Wen et al. [25] compare a question to configuration option names based on this equation:

$$sim(q;o) = \sum weight_{t_i;o} \quad occ(t_i;q) \tag{5}$$

where *occ* refers to the number of occurrences of a term in the input question.

Wen et al. [25] considers the title and the body of a bug report (config-related question) so they can maximize the chances of finding the appropriate option, while Jin et al. [10] used just a config-related questions title and extended it with the synonyms of each word of that title by using WordNet [15]. Besides, Jin et al. [10] scale down the impact of these synonyms by slightly changing the previous equation as follow:

$$sim(q;o) = \sum scale \quad weight_{t_i;o} \quad occ(t_i;q) \tag{6}$$

where *scale = 1* for the original words of the question and *scale = 0.4* for synonyms of these words. Jin et al. [10] have explored different values for that scale and found that scale of 0.4 reports the best results.

Finally, we sort options based on the textual similarity between each of these options' names and the question.

5.2 Google Based Search

In addition to the previous two baseline approaches, we also compare ConfigMiner to a basic Google search. We simulate a user asking his question by writing a couple of keywords on Google. We use for each of the 2,046 config-related questions the following keywords for Google searches:

The title of each question.
The first 15 non-stop and unique keywords.
The first 32 non-stop and unique keywords. Note that we cannot use the whole question since Google search has a limit of 32 keywords.

We limit our search results to StackOverflow for each of our seven case studies, while we use StackOverflow and Firefox forum websites for Firefox config-related questions. We select the top 20 search results that do not point to the question itself.

Once the StackOverflow and Firefox forum links are obtained, we crawl Firefox pages and extract StackOverflow questions. Then, we associate to each obtained question its appropriate options using the same approach of Section 3.1. Note that the rank of an option corresponds to the rank at which Google reports the link to the StackOverflow (or Firefox forum) thread whose accepted answer mentions that option. For example, an option has a rank of 3 if Google reports its associated StackOverflow or Firefox forum link at rank 3.

6 Empirical Evaluation

We evaluate ConfigMiner on the 2,062 config-related questions (**Dataset 1**) that are shown in Table 1. For our evaluation, we used the preprocessed title of a question and its words synonyms for PrefFinder while we use the preprocessed title and body of questions for CoLUA and ConfigMiner. In addition, we evaluate ConfigMiner and the baseline approaches on the 275 config-related questions (**Dataset 2**) that we were able to manually classify into one of three types of questions (Section 3.2). We structured our evaluation using the following research questions:

RQ1 How accurate is ConfigMiner compared to the baseline approaches on identifying at least one appropriate option?

RQ2 How much does ConfigMiner's accuracy vary across the three types of config-related questions?

RQ3 How much time is ConfigMiner likely to save?

6.1 Results:

RQ1 How accurate is ConfigMiner compared to the baseline approaches on identifying at least one appropriate option?

Motivation: We aim to evaluate the effectiveness of ConfigMiner in identifying at least one appropriate compared to our baseline approaches.

Approach: Our evaluation consists of identifying how many questions (in Dataset 1) for which ConfigMiner is able to identify the appropriate options. Since our approach assumes the existence of a database of already-answered config-related questions, we treat, in our evaluation, each of the already-answered config-related questions of a case study as a new question and calculates its similarity to the rest of the questions using the ConfigMiner approach. For example, for each of the 789 Firefox questions, our approach calculates its similarity to each of the remaining 788 questions to recommend their respective solutions. Concretely, we evaluate our approach using the following metrics:

- Accuracy at 1, 5, 10, and 20: We first calculate the percentage of questions for which ConfigMiner and the baseline approaches identify at least one appropriate option within the top 5 identified options (i.e, accuracy at 5). For example, ConfigMiner can have an accuracy at 5 of 80% if it is able to identify one appropriate option for 80% of the config-related questions within the top-5 identified options. We similarly obtain the accuracy at 1, 10, and 20.

- *Bootstrap sampling:* We calculate the accuracy at 1, 5, 10, and 20 on 2,000 different bootstrap samples to mitigate the errors within the 2,062 config-related questions and to produce statistically robust conclusions similar to prior work [20], [22], since bootstrap resampling produces more accurate estimates [23]. The errors within our dataset consist of questions that are not related to configuration but that were accidentally selected using the approach of Section 3. To mitigate this problem, we calculate the accuracy of ConfigMiner and the baseline approaches using 2,000 bootstrap samples. We use the following steps:

> We select a bootstrap sample whose size is equal to the number of questions of a given case study (e.g., a bootstrap sample of 789 Firefox questions).
> We calculate the accuracy at 1, 5, 10, and 20 of that obtained sample using Google search, PrefFinder, CoLUA, and ConfigMiner.
> We repeat the same process 2,000 times to obtain at the end an accuracy distribution, for which we report the median of the accuracies obtained.

We measure the overall improvement of our approach over the baseline approaches using the following equation:

$$\frac{\text{The median accuracy of ConfigMiner}}{\text{The median accuracy of the best performing baseline approach}} - 1 \quad (7)$$

- Rank at which the appropriate options are reported: While the previous accuracy measure evaluates the number of questions for which we are able to identify the appropriate option within the top 1, 5, 10, and 20 identified options, we also evaluate the ranks at which these appropriate options are reported. If an already-answered question has more than one appropriate option, ConfigMiner reports all of them in a random order. Note that we do not report the rank at which the Google approach identifies an option since our search is limited to just the top 20 websites.

To mitigate the risk of errors on config-related questions, we also followed a bootstrap approach to calculate the ranks. Mainly, we use the following steps:

> We select a bootstrap sample whose size is equal to the number of a case study questions.
> We calculate the statistically significant difference between the ranks that ConfigMiner, PrefFinder, and CoLUA found (using Wilcoxon test). We also calculate the median rank obtained by each of the three approaches.
> We repeat the same process 2,000 times.
> We see if our findings are consistent by counting how often the differences between the rank that are reported by the three approaches were significant.

Results: ConfigMiner outperforms prior approaches by a median of 130% (for the accuracy at 20) for identifying at least one appropriate option for a config-related question. As shown in Table 2, ConfigMiner outperforms the baseline approaches on the evaluated accuracy at 1, 5, 10, and 20. Furthermore, ConfigMiner is statistically more accurate compared to the baseline approaches on all the config-related questions of the 7 case studies (Wilcoxon test; p-values < 2.2e-16; = 0.01). Note that we calculated the statistically significant differences between the accuracy distribution that we obtained using bootstrap samples.

ConfigMiner significantly improves the rank at which the appropriate options are reported for the questions that are related to five and six out of seven software systems compared to PrefFinder and CoLUA respectively. While we did not find any case study for which one of the two baseline approaches outperforms ConfigMiner, we were not able to improve the rank for Cassandra and MapReduce, as shown in Table 3, although ConfigMiner is still more accurate compared to both baselines on these case studies.

ConfigMiner reports the appropriate options at a median rank of 4, compared to a median rank of 17 and 20 that are reported by PrefFinder and CoLUA. As shown in Table 4, our approach improves the median rank from 16 to 4 and from 111.5 and 75 to just 5 for Yarn questions compared to CoLUA and PrefFinder respectively.

> ConfigMiner is more accurate compared to the baseline approaches and reports options at a lower median rank of just 4.

RQ2 How much does ConfigMiner's accuracy vary across the three types of config-related questions?

Motivation: The goal of this research question is to evaluate if the accuracy of ConfigMiner is consistent through the three types of config-related questions.

Approach: We evaluate the effectiveness of ConfigMiner and the baseline approaches on the 275 config-related questions (Dataset 2) that we manually classified as one of three

TABLE 2
The Accuracy of ConfigMiner Compared to the Baseline Approaches. Note that we report just on the best Google search strategy.

	Accuracy at 1				Accuracy at 5				Accuracy at 10				Accuracy at 20			
	Pref-Finder	CoLUA	Google	Config-Miner	Pref-Finder	CoLUA	Google	Config-Miner	Pref-Finder	CoLUA	Google	Config-Miner	Pref-Finder	CoLUA	Google	Config-Miner
Firefox	4%	5%	6%	**35%**	10%	12%	16%	**59%**	14%	16%	19%	**69%**	18%	22%	23%	**76%**
HDFS	6%	12%	1%	**28%**	15%	25%	6%	**61%**	18%	32%	8%	**73%**	22%	40%	14%	**79%**
Yarn	4%	5%	8%	**20%**	10%	18%	15%	**58%**	13%	24%	20%	**73%**	18%	36%	28%	**85%**
Hadoop	1%	2%	1%	**34%**	6%	10%	7%	**85%**	7%	14%	11%	**89%**	9%	19%	16%	**91%**
Spark	9%	14%	3%	**20%**	23%	36%	12%	**57%**	33%	49%	17%	**72%**	46%	63%	23%	**83%**
MapReduce	11%	16%	3%	**22%**	23%	32%	7%	**53%**	30%	44%	9%	**68%**	33%	54%	16%	**80%**
Cassandra	4%	8%	4%	**32%**	12%	18%	12%	**66%**	13%	25%	15%	**76%**	14%	32%	19%	**88%**
Median	4%	8%	3%	**28%**	12%	18%	15%	**59%**	14%	25%	15%	**73%**	18%	36%	19%	**83%**

TABLE 3
The percentage of bootstrap samples that shows a statistically significant differences (using Wilcoxon test) between ConfigMiner and each of the two baseline approaches. Note that we did not observe in any of the 14,000 samples that one of the baseline approaches outperforms ConfigMiner's ranking results.

	ConfigMiner vs PrefFinder	ConfigMiner vs CoLUA
Cassandra	0.8%	49%
Firefox	100%	100%
Hadoop	100%	100%
HDFS	96%	100%
MapReduce	47%	100%
Spark	100%	100%
Yarn	100%	100%

TABLE 4
Median rank at which appropriate options are reported.

	PrefFinder	CoLUA	ConfigMiner
Cassandra	3	6	4
Firefox	16	24	**3**
Hadoop	29	41	**2**
HDFS	8.5	20.5	**3**
MapReduce	7	12	**5**
Spark	22	15	**6**
Yarn	111.5	75	**5**
Median	16	20.5	**4**

types of config-related questions: questions about how to fix an error with an explicit error message, questions on how to fix errors without any explicit message, and how-to-do type of questions. Similarly to RQ1, we evaluate ConfigMiner, PrefFinder, CoLUA, and a Google based search using the 4 types of accuracy: Accuracy at 1, 5, 10, and 20.

Results: ConfigMiner outperfrorms the baseline approaches on identifying the appropriate options for all of the three types of config-related questions. We found that ConfigMiner outperforms the baseline approaches to find the appropriate options for questions that ask about how to fix an error with an explicit message, an error without explicit message, and how-to-do type of questions (as shown in Table 1, Table 2, and Table 3 in the Appendix). That improvement ranges between 64% and 228% for the accuracy at 20 and compared to CoLUA, whose results outperform PrefFinder and Google search's accuracy at 20.

The rank at which the appropriate options are reported is consistent across the three types of config-related questions, as shown in Figure 7. Indeed, we did not find any

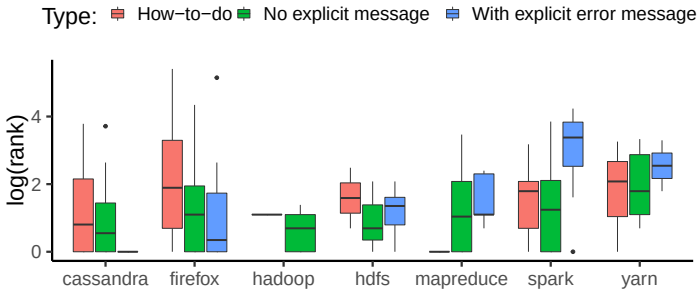

Fig. 7. Rank (In log scale) at which ConfigMiner reports the appropriate options for each of the 3 categories of questions.

statistically significant differences between the three types of config-related questions related to each of the 7 case studies.

> ConfigMiner accuracy and ranks are consistent through the three types of config-related questions.

RQ3 How much time is ConfigMiner likely to save?

Motivation: The goal of this research question is to evaluate the amount of time that is required by users on each identified configuration option by ConfigMiner until finding the appropriate one, such that ConfigMiner still outperforms asking the question in an online forum.

Approach: We used the "break even (BE)" metric that was proposed by Jin et al. [10]. It consists of measuring how much time does a user need to spend on investigating on each reported option (e.g., finding the possible values or understanding the descriptions) before finding the appropriate option. BE is defined as:

$$BE = \frac{\text{Time to receive the accepted answer}}{\text{Rank of the appropriate option}} \quad (8)$$

Similarly to RQ1, we calculate the BE over bootstrap samples to mitigate the risk of errors on our dataset of questions. Concretely, we follow these steps:

We select a bootstrap sample similarly to RQ1.
We calculate the BE metric for each question in the sample, and we select the median BE.
We repeat the same process 2,000 times. Then, we report on the median of the whole samples.

To better understand the efforts required to type a whole question, we measure the textual length of our evaluated questions and their association with the rank at which ConfigMiner reports the appropriate option. Note that we use Dataset 1 questions in this research question.

TABLE 5
The break even metric for questions for which the appropriate option is reported at each of the following rank range. For example, if a user spend 95 minutes on each option before finding the appropriate option (which is reported at a rank between 2 and 5), our approach is still better than asking the question online.

	Any rank	[1,1]	[2,5]	[6,10]	[11,20]
Cassandra	119	317	95	35	28
Firefox	52	115	86	20	31
Hadoop	172	250	159	10	44
HDFS	131	235	137	53	194
MapReduce	109	180	213	26	91
Spark	69	307	93	49	31
Yarn	149	833	396	84	97
Median	119	250	137	35	44

Results: Even if a user spend a median of 119 minutes on each reported option by ConfigMiner, our approach can be still valuable, as shown in Table 5. For example, if a user spend a median of 35 minutes on each option until finding the appropriate option that is reported at a rank that ranges between 6 and 10, our approach can still outperform asking the question in an online forum. Note that our analysis does not consider options that ConfigMiner is not able to report at any rank (7%). In addition, our analysis considers just the delay required to receive the accepted answer in an online forum, while users still need to spend further efforts on that accepted answer, such as understanding the answer or testing the suggested options.

Finally, there is no correlation (Spearman correlation of 0.02) between the length of a config-related question and the rank at which the appropriate option is reported by ConfigMiner. Our evaluated config-related questions have a median of 85 words (the source code snippets are not considered), out of which 41 are just stop-words. We also observe that ConfigMiner reports the appropriate option with a median rank of 2 for config-related questions that are less than 20 non-stop words. Therefore, we believe that typing a median of 44 non-stop words is more practical than waiting for an answer in StackOverflow for a median of 4.75 hours (as discussed in Section 3.3).

While the goal of our paper is to show that using the already-answered questions is more accurate than textually comparing a question to option names, future work should investigate how to improve the usability of ConfigMiner (e.g., how to leverage the already-answered config-related questions to provide a richer context to the suggested option). In addition, the documentation of a suggested option can be shown to help users better understand the suggested options, as proposed by Jin et al. [10].

Even if a user spend a median of 119 minutes on each option until finding the appropriate option, ConfigMiner is still better than asking a question in an online forum and waiting for the answer.

7 Discussion

While ConfigMiner is not able to identify the appropriate options for 7% of the config-related questions, the baseline approaches missed a median of 67% and 29% of the config-related questions. ConfigMiner is not able to identify options if they were never previously appropriate for any already-answered question, the number of such questions that ConfigMiner cannot address is 7% from the 17% of config-related questions for which ConfigMiner is not able to identify the appropriate option within the top-20. The accuracy of our approach might be improved by expanding our data set of config-related questions using other data sources (e.g., mailing lists or other forums).

A config-related question that is not resolved by the baseline approaches:

Title: No 4K resulution on youtube
Body: *Some videos on youtube don't show the 4k resolution option under firefox and the only go up to 1080P but on chrome they show all the resolutions:*
example video https://youtu.be/WU-NrfB8Kd4
I have flash installed as well

***The appropriate option**: media.mediasource.webm.enabled*

Fig. 8. A real example of a user-question that was not resolved by the baseline approaches.

The baseline approaches cannot find an appropriate option for a config-related question if they do not share any word with the question's title and body. The appropriate options for 67% and 29% config-related questions are not reported at any rank (even higher than 20) by PrefFinder and CoLUA respectively. The titles of the 67% config-related questions as well as their synonyms do not share any words with their appropriate options. Similarly, the title and body of the 29% config-related questions that are missed by CoLUA do not share any word with their appropriate options. Thus, the similarity between each of these missed config-related questions and their appropriate options is equal to zero. Figure 8 shows a concrete example of a config-related question that is missed by both baseline approaches, as neither the title nor the body share any word with the appropriate option *"media.mediasource.webm.enabled"*, while ConfigMiner identifies that option at the first rank.

The baseline approaches are not accurate on the config-related questions for which ConfigMiner did not find the appropriate options. For the 7% of config-related questions for which ConfigMiner is not able to identify the appropriate options, PrefFinder and CoLUA show an accuracy at 20 of just 28% and 42%.

Therefore, we believe that a direct combination of our approach and the baseline approaches might not be able to improve the accuracy of identifying the appropriate options. While extending our database of already-answered questions might improve the accuracy of our approach, the baseline approaches might be more appropriate for questions related to a new software system. While the most straightforward way to enrich that database is to use online forums, we can still leverage other sources such as bug reports, mailing lists, or relying on users feedback on ConfigMiner suggestions. Furthermore, even with few occurrences of an option in our database we are still able to identify the appropriate option since we observe a weak Spearman correlation (-0.23) between the popularity of an option (the number of questions for which an option is

appropriate) and the rank at which ConfigMiner reports it.

The effectiveness of the baseline approaches is also penalized by the number of words that a config-related question shares with false positive options. The baseline approaches identify the appropriate options for a median of 15% and 35% of the config-related questions outside the top-20 identified options because these questions share more words with false positive options than the appropriate options. For the same reason, PrefFinder is not able to find the appropriate options for any Yarn and Hadoop config-related questions within the top-10, while it is able to identify the appropriate options for 20% of Yarn config-related questions until the top-20 identified list of options (Table 2 in the Appendix). For example, PrefFinder reports the appropriate option for a Yarn config-related question at the 228th rank, because that question and its appropriate option share just the word *"yarn"*. Furthermore, that *"yarn"* word has a low TF-IDF weight as it is frequently used by Yarn's configuration options (396 out of 397 Yarn options use that word), which makes the similarity score between that question and its appropriate option low compared to the other 227 options that share more words with the question. Note that ConfigMiner is able to identify that option at the second rank.

8 Threats to Validity

A threat to external validity concerns the generalizability of our results. Our evaluation considers 7 case studies and questions from three forums. We cannot generalize our results to other systems nor to other forums. However, our studied software systems are popular and have a large number of configuration options. We do not consider bug reports since our approach requires a mapping between each bug report and its appropriate option, which we cannot automatically identify. However, our evaluation shows motivating results that can be replicated on bug reports.

An internal threat to validity concerns the collection of our evaluated questions. As we automatically collected the evaluated questions using the approach of Section 3, some of the obtained questions might not be related to configuration. To mitigate this risk, we used four different heuristics that show a good accuracy of 92% 5% and we evaluate our approach on a large number of bootstrap samples. Our data selection might also miss config-related questions for which the accepted answer misspells a configuration option name. Future work should consider additional heuristics to identify more config-related questions.

A second internal threat to validity concerns the completeness of the list of options that we used. We used the list of options that are mentioned in the documentation of the studied systems, since such a list is the official source of options, although it might also be incomplete and outdated. Future work should investigate other sources of options.

Another internal threat to validity concerns the misclassification of an option as appropriate for a question, which can influence our evaluation. For example, other answers than the accepted one might mention better solutions (aka more appropriate options). To mitigate this risk, we consider just the options that are mentioned in the accepted answer since it is supposed to be the best solution for a question[8]. Future work should identify more heuristics to identify the most appropriate option for a question.

A final internal threat to validity concerns the qualitative analysis. Although the process might be subjective and represents a threat to internal validity, it was a straightforward (but time-consuming) task since we had to classify each question into three distinct high-level categories of questions. Therefore, we did not cross-validate our manual analysis.

Prior work on recommending configuration options [10], [33] as well as our work have the same threats of knowing first whether a question is related to configuration or not. Prior work by Bowen et al. [28], Wen et al. [25], and Xia et al. [26] tackle this pre-step by determining whether a bug report is related to a configuration, while prior approach address the following step of identifying the appropriate values of a configuration option by Xiong et al. [27].

9 Conclusion

This paper studies the different types of config-related questions and proposes an automated approach, ConfigMiner, to identify their appropriate configuration options. We observe that users ask three types of config-related questions: they ask about configuration options to fix an error with an explicit message, an error without any explicit message, and how-to-do type of questions. While the first type of config-related questions are addressed by a large body of prior work, few approaches can address the second and third types of config-related questions, even if they are as important as configuration errors with an explicit message since these two types of config-related questions are common (56% of config-related questions) and require a similar amount of time to be correctly answered.

Two prior approaches exist to identify the appropriate options for a config-related questions [10], [25]. They rely on the textual similarity between a config-related question and the option names. However, config-related questions do not necessarily share words with their appropriate options.

Consequently, we propose an approach that leverages already-answered config-related questions and their associated answers to identify the appropriate configuration options for new question. Our empirical evaluation shows that ConfigMiner is more accurate compared to the two baseline approaches. In fact, ConfigMiner improves the state-of-the-art approaches by a median of 130%. Finally, using the history of config-related questions shows better results than using just option names.

References

[1] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K., Schneider, K.A.: Mining duplicate questions in stack overflow. In: Proceedings of the 13th International Conference on Mining Software Repositories, pp. 402–412. ACM (2016)

[2] Andrzejak, A., Friedrich, G., Wotawa, F.: Software configuration diagnosis-a survey of existing methods and open challenges. In: ConfWS, pp. 85–92 (2018)

[3] Attariyan, M., Flinn, J.: Automating configuration troubleshooting with dynamic information flow analysis. In: Proceedings of the 9th Usenix Symposium on Operating Systems Design and Implementation (OSDI), pp. 1–14 (2010)

8. https://stackoverflow.com/help/someone-answers