Mining Software Information Sites to Recommend Cross-Language Analogical Libraries

Kawser Wazed Nafi, Muhammad Asaduzzaman†, Banani Roy, Chanchal K. Roy and Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Canada
†Department of Computer Science, Lakehead University, Canada
{kawser.nafi, banani.roy, chanchal.roy, kevin.schneider}@usask.ca
†md.asaduzzaman@lakeheadu.ca

Abstract—Software development is largely dependent on libraries to reuse existing functionalities instead of reinventing the wheel. Software developers often need to find analogical libraries (libraries similar to ones they are already familiar with) as an analogical library may offer improved or additional features. Developers also need to search for analogical libraries across programming languages when developing applications in different languages or for different platforms. However, manually searching for analogical libraries is a time-consuming and difficult task. This paper presents a technique, called XLibRec, that recommends analogical libraries across different programming languages. XLibRec collects Stack Overflow question titles containing library names, library usage information from Stack Overflow posts, and library descriptions from a third-party website, Libraries.io. We generate word-vectors for each information and calculate a weight-based cosine similarity score from them to recommend analogical libraries. We performed an extensive evaluation using a large number of analogical libraries across four different programming languages. Results from our evaluation show that the proposed technique can recommend cross-language analogical libraries with great accuracy. The precision for the Top-3 recommendations ranges from 62-81% and has achieved 8-45% higher precision than the state-of-the-art technique.

Index Terms—Cross-Language, Analogical Libraries, Stack Overflow, Library Usage Information, Word2Vec

I. INTRODUCTION

Developers extensively depend on software libraries to reuse existing functionalities instead of working from scratch. This saves both development time and effort [1]. As a result, libraries are an integral part of modern software development [2], [3]. For example, Thung et al. found that 93.3% of the software projects they selected from GitHub use third-party libraries at an average rate of 28 libraries per project [2]. Thus, it is important to select a suitable library to complete a task when developing a software application. Often developers need to replace an existing library with a different one. This is perhaps because the library is no longer under development or lacks certain functionality that is offered by another library. It may also be the case that a developer is looking for a library similar to an existing one but written in a different language to make their software available to a large group of users. However, finding such cross-language analogical libraries is a non-trivial task.

One possible way to find cross-language analogical libraries is to search in forums, blogs or online tutorials. However, manually doing so is a time-consuming operation. Figure 1 shows an example of a Stack Overflow post where a developer asks for an analogical library in Python for the HttpClient library in Java. The search results need to be checked manually to find the library (for this case, requests library). Many of such searches may not lead to correct results due to vocabulary mismatch problems [4]. To address such limitations, Chen et al. [5], [6] developed a tool, called SimilarTech [7], that can recommend analogical libraries by mining Q&A discussions in Stack Overflow, a community question answering site specifically developed for discussing programming related questions. Their technique considers the tags of a Stack Overflow question as a tag sentence where each tag represents a word in that sentence. The basic idea of the technique is

Fig. 1: An example of a Stack Overflow post
that analogical libraries would share similar context in their tag sentences. While the technique is promising, it requires library names to be used as tags in Stack Overflow questions. However, during our manual study we observe that many popular libraries are not used as tags in Stack Overflow questions, such as FastNLP and geopython in Python, simple-react and sharpremote in C#/.NET, and so on. Furthermore, two analogical libraries may not share similar context in their tag sentences. These issues can impact the performance of the technique.

To address the above mentioned issues, this paper presents a technique, called XLibRec, to recommend cross-language analogical libraries. This technique depends on two different sources of information. First, we mine Stack Overflow (SO) posts (question titles and answer bodies) to determine library usage information (LUI) associated to the mentioned libraries. Second, we collect descriptions of libraries from the Libraries.io (LibIO) data dump [8]. We collect both Verbs and Nouns those appeared in the close proximity of library names mentioned in both SO posts and LibIO descriptions.

Our technique is based on the hypothesis that analogical libraries should be associated with similar library usage information in both of the sources. We mined all these LUIs with the help of Mikolov et al.’s Word2Vec [9] method and calculated similarity scores among the collected LUIs. Finally, we aggregated all these similarity scores with a weight-based mechanism to recommend cross-language analogical libraries. To evaluate our proposed technique, we performed a manual study on the recommendation of 900 randomly selected libraries across four different programming languages (Java, Python, C#, and JavaScript) with the help of eight carefully chosen software developers. From our study we found that XLibRec can recommend analogical libraries across different programming languages with an average precision of 0.70 for the Top-1 recommendations, 0.74 for the Top-3 recommendations and 0.72 for the Top-5 recommendations which outperform the existing state-of-the-art technique (i.e., SimilarTech [7]). We have made the tool available to support future research in Cross-Language software development.

Thus, the contributions of the paper are as follows.

- A technique that combines information from two different sources to recommend cross-language analogical libraries.
- Evaluation of the proposed technique against the state-of-the-art technique.
- Additional analyses to understand the benefits of the proposed technique.

The remainder of the paper is organized as follows. Section II describes our proposed technique. Section III explains the evaluation procedure. Section IV discusses our results and presents additional analyses on the proposed technique. Section V summarizes the threats to the validity of this study. Section VI discusses related work to our study. Finally, Section VII concludes the paper.

## II. PROPOSED TECHNIQUE

This section discusses our proposed technique XLibRec for recommending cross-language analogical libraries. XLibRec works in the following three steps as shown in Figure 3:

- **Mining Library Usage Information (LUI):** We collect three types of LUI from two different sources: i) Stack Overflow (SO), ii) Libraries.io (LibIO). The LUIs are: a. Words appeared around a Library name in SO posts, b. Question titles related to a library, c. Library description from LibIO. After collecting them we apply Word2Vec on each of them individually to generate vectors.

- **Score Calculation:** Next, we calculate Cosine similarity among the vectors collected in the previous step in respect to their LUIs for every probable cross-language analogical library pairs. For each pair, we get three Cosine similarity scores. Finally, we aggregate them in a weight-based manner to generate the final similarity score for a pair.

- **Analogical Library Recommendation:** For a source library, we recommend the Top-k highly similar cross-language analogical libraries based on the previously calculated aggregated scores. The cross-language analogical libraries are recommended in descending order of their similarity scores.

In the following, we are going to describe each step in detail.

### A. Mining Library Usage Information (LUI)

This section discusses how we mine library usage information from SO and LibIO.

1) **Selection of questions and answer bodies from SO:** Mining SO is a difficult task due to the presence of a large number of questions and answers. Many of those questions and answers discuss problems experienced while using or configuring libraries, such as an error regarding the Jackson library in the post [https://stackoverflow.com/questions/23720251/getting-error-in-jackson-library-code?](https://stackoverflow.com/questions/23720251/getting-error-in-jackson-library-code?). To filter out these irrelevant SO posts and to capture the related ones, we followed an approach proposed by Tredeu et al. [10]. Let us consider the example given in Figure 4 where a developer asked for a library recommendation to remove HTML tags from a string in Java. Developers recommended libraries which are JTDY, JERICHO and HMLCLEANER. With a closer look we found that developers used some keywords (such as check out, using and can use), at the time of recommending libraries. This motivates us to search for patterns of recommending libraries which will allow us to identify library posts where LUI regarding a library was discussed. We collected library names from two sources, library tags from SO questions (SO tags that refer to library names) and library names from LibIO which we discussed in Section III-A. We searched for SO posts where those library names had appeared. We then

[https://github.com/Kawser-nerd/XLibRec](https://github.com/Kawser-nerd/XLibRec)  
[https://stackoverflow.com/questions/240546](https://stackoverflow.com/questions/240546)
TABLE I: Patterns of recommending library names

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Question ID</th>
<th>Pattern Example in Postbody</th>
<th>Question Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>use</td>
<td>6332593</td>
<td>Also you can use XPath to navigate through XML.</td>
<td>Processing XML in Java</td>
</tr>
<tr>
<td>check</td>
<td>285546</td>
<td>You can also check if Id</td>
<td>Remove HTML tags from a String</td>
</tr>
<tr>
<td>try</td>
<td>473528</td>
<td>If you for any reason use pre 1.5 Java then may</td>
<td>How can I pad an integer with zeros on the left?</td>
</tr>
<tr>
<td>using</td>
<td>6853972</td>
<td>Using Gson, you can define classes.</td>
<td>Getting Data from JSON</td>
</tr>
<tr>
<td>good solution</td>
<td>1324567</td>
<td>Apache Cocoon is a very good solution</td>
<td>Implementing RESTful Web Services.</td>
</tr>
<tr>
<td>good alternative</td>
<td>34726</td>
<td>Itrun development has pretty much stopped. You should look into running</td>
<td>Trouble using JRun to Host Java Servlets</td>
</tr>
<tr>
<td>looked into</td>
<td>1816550</td>
<td>Have you looked into Clojure? It's a Lisp dialect that runs on the Java Virtual Machine</td>
<td>Lisp code called from Java</td>
</tr>
<tr>
<td>similar library</td>
<td>3895718</td>
<td>You can use commons-gson or similar library to convert the object into json</td>
<td>Object oriented design pattern for parsing json files</td>
</tr>
<tr>
<td>give a try</td>
<td>7503036</td>
<td>you can give a try to JSON</td>
<td>How to search in not stricted HTML with java?</td>
</tr>
<tr>
<td>try out</td>
<td>1272090</td>
<td>You can try out the Gxt Gantt, built on the Ext GWT framework. Gxt Gantt 2.0</td>
<td>How to develop GWT widget?</td>
</tr>
<tr>
<td>looking for</td>
<td>8638502</td>
<td>Or maybe Scala, Clojure or Groovy are what you are looking for</td>
<td>Filtering a list of JavaBeans with Google Guava</td>
</tr>
<tr>
<td>suggest to</td>
<td>8563294</td>
<td>I could suggest to use Apache Commons IO library.</td>
<td>Modifying existing file content in Java</td>
</tr>
</tbody>
</table>

![Fig. 2: An example of a SO question to complete a task and answerers recommend three different libraries](https://libraries.io/data)

manually analyzed randomly selected 5K Stack Overflow posts (questions and their associated answers) and we were able to identify 12 patterns of recommending library names by developers, as shown in Table I. If any of those 12 patterns appeared in an answer of a SO post, we collected the complete post (question and all associated answers) and considered that as a LUI source.

2) Mining LUI from SO Answer Body: Once we collected all the posts from the previous step, we filtered out all the coding parts, hyperlinks and only kept the paragraph-text for every post. Next, we used Python NLTK-PoS [11], [12] tagger to tag Verbs and Nouns of each paragraph in the SO posts. We also used stemming of NLTK [13] to get the original verbs and removed to-be verbs from the tagged list. For example, for the Jsoup library mentioned in Figure 2, we considered Verb and Noun words (i.e., USE, HTML, PARSER, REGEX, HTML, TAGS, and WANT). We followed the procedure for all libraries and their related posts. After this, we considered the list of Nouns and Verbs for each library as a Bag-of-Words and used T. Mikolov’s [9] pretrained Word2Vec Google model to generate word vectors for each library. Finally, we used Cosine similarity [14] to detect the similarity between two libraries using the following equation:

\[ PB_{Sim} = \cos(A, B) \]  

where \( A \) and \( B \) are the word vectors of Library A and B, respectively.

3) Mining LUI from SO Question Title: Developers often ask questions in SO seeking help to solve a problem which they state in question titles. In response, other developers suggest library names or code examples that can solve the problem. This often states the functionality of those libraries. For example, a developer asked help for removing HTML tags from a String in Java which is clearly depicted in the question title (see figure 2). In response to this question, other developers suggest three libraries (“JTIDY”, “JERICO”, “HTMLCLEANER”) that can solve the problem. From this example we can say question titles are capable to capture the functionalities of different libraries. Thus, we considered Question titles as a source of LUI regarding a library. We first collected all the question titles for a library where that library name appeared in answer bodies. We performed this step for all the library names we considered. Next, we applied NLTK-PoS tagging along with stemming to the question titles. From the resulting tags we only kept Noun and Verb words for each library. Finally, we applied Word2Vec for generating vectors, aggregated them and calculated the Cosine similarity.

\[ QT_{Sim} = \cos(A, B) \]  

where \( A \) and \( B \) are the word vectors of Library A and B, respectively.

4) Mining LUI from library description: In SO, developers seek help on how to use a library. There is a high probability that this library is not used as a library tag (i.e. library name that appears as a tag) in SO. To leverage this gap, we used another LUI source, Libraries.io from which we collected library names and their descriptions. Libraries.io collects libraries from 37 package managers for different programming languages.

Collecting library names from Libraries.io is a challenging task. Most of the libraries are collected with their package
names. We filtered out the package names and only kept the library names. For example, one library name in Java is `com.datumbox:datumbox-framework`. We select the text to the right of the colon (‘:’), tokenize it and extract the first word “datumbox”. We do the same for other library names too. For each library, we collected the short descriptions provided by the developers. Later, we applied Word2Vec to generate word vectors from descriptions and Cosine similarity to calculate their similarity score using following equation:

\[
\text{LibIOSim} = \cosine(\bar{A}, \bar{B})
\]

where \(\bar{A}\) and \(\bar{B}\) are the word-vectors of library descriptions for the library A and B, respectively.

### B. Similarity Score Calculation

Figure 3 shows an overview of the workflow of XLibRec. For our study, we considered four programming languages (i.e. Java, C#, Python and JavaScript). At step 1, two libraries are selected from two different programming languages. Next, we collect the LUI for those libraries from SO posts as discussed in Section II-A2 (using steps 2 and 3). Using steps 5 and 6 we collect LUI from the question titles as discussed in section II-A3. Using step 8 we collect the library descriptions for those two libraries from Libraries.io as discussed in Section II-A4. We then use Word2Vec pre-trained model\(^\text{6}\) to generate word-vectors for the LUIs collected using previous steps and Cosine similarity to calculate similarity score among the vectors (steps 4, 7 and 9). Once the similarity scores are calculated for all the three information sources between those two libraries selected in step 1, we use the following equation to calculate an aggregate final similarity score (ranges between 0 and 1):

\[
\text{LibSim}(A,B) = \alpha \times PB_{Sim}(A,B) + \beta \times QT_{Sim}(A,B) + \gamma \times \text{LibIO}_{Sim}(A,B)
\]

\(^6\)https://code.google.com/archive/p/word2vec/

where A and B are two libraries of two different programming languages. For this equation, values of \(\alpha\), \(\beta\) and \(\gamma\) are calculated using the Adaptive Hill Climbing algorithm\(^\text{15}\). For our technique we found that the most suitable values for \(\alpha\), \(\beta\) and \(\gamma\) are 0.30, 0.35 and 0.35, respectively. We repeat the process for all the probable analogical library pairs. We store the scores along with the probable analogical library names using a similarity threshold of 0.6.

### C. Analogical Library Recommendation

When a developer is looking for cross-language analogical libraries of a library, they can interact with the XLibRec interface (step 14). The developer needs to provide the library name (step 15) for which they are looking for analogical libraries. At step 16, analogical libraries are retrieved along with their similarity scores for the given library. At step 17, a recommendation list is created using the Top-5 analogical libraries based on their similarity scores. Finally, this list is returned to the developer from which they can select the most suitable analogous library for their work (step 18).

## III. EVALUATION

We evaluate the performance of our proposed technique in two different ways. First, we compare the technique with SimilarTech\(^\text{7}\), a state-of-the-art technique for recommending cross-language analogical libraries. Second, we also compare the technique with Google search\(^\text{16}\) to understand the benefit of using XLibRec instead of searching on the web.

### A. Dataset Overview

We collect three different datasets, each consists of a set of libraries, to evaluate the performance of techniques for recommending analogical libraries. A brief overview of each dataset is given below.

1. **Dataset-A**: We collected the set of libraries for which SimilarTech determined analogical libraries\(^\text{17}\). We obtained 467 Java, 426 Python, 372 C# and 405 JavaScript libraries. The...
collection of such libraries represents the Dataset-A. SimilarTech generated the knowledge base of analogical libraries using Stack Overflow post data from July 31, 2008 to August 16, 2015. For the purpose of evaluation, we randomly selected 200 libraries, 50 for each language. We refer to the set of libraries as Dataset-A' and performed our evaluation on those libraries.

2) Dataset-B: We downloaded a publicly available data dump of Stack Overflow from archive.org which contains all the data for Stack Overflow from July 31, 2008, to January 20, 2020 and collected library tags. Next, we filtered out libraries considered in dataset-A to form Dataset-B to see whether SimilarTech performs the same or not. The dataset contains 328 Java, 358 Python, 285 C# and 302 JavaScript libraries. For our evaluation, we randomly selected 300 libraries, 75 for each language, out of those libraries. We refer to the set of libraries as Dataset-B'.

3) Dataset-C: This represents a set of libraries that are not used as tags in SO questions. We collected the dataset as follows. We used the “star-count” and “source-rank” properties of each library to select the top 1500 libraries from LibIO for each of the four languages (i.e., Java, C#, Python and JavaScript). Next, we filtered out those libraries which are not present in Dataset-A and Dataset-B. The corresponding set of libraries represents Dataset-C which consists of 1032 Java, 925 Python, 981 C# and 1089 JavaScript libraries. These libraries are not used as tags in SO. For the purpose of evaluation, we randomly selected 400 libraries, 100 for each of the languages. We refer to the set of libraries as Dataset-C'.

B. Research Questions

To evaluate XLibRec, we answer the following three research questions.

RQ1 How effective is XLibRec in recommending analogical libraries for Chen et al.'s dataset (i.e., Dataset-A)?

RQ2 How effective is XLibRec in recommending analogical libraries for Dataset-B’?

RQ3 How effective is XLibRec in recommending analogical libraries for Dataset-C’?

C. Experimental Setup & Validation Strategy

To validate the performance of our technique, we are required to manually check each of the library pairs of four programming languages recommended by our technique. We queried with a library name of one programming language as the source and recorded all the recommended analogical libraries in other languages (i.e., destination). We could experiment with all the combinations of four languages as ⟨source, destination⟩ pairs. However, it would be time consuming, costly, infeasible and would require extensive man power. To keep our validation process feasible, without evaluating all possible language pairs, we randomly selected five ⟨source, destination⟩ combinations: ⟨Java, Python⟩, ⟨Java, C#⟩, ⟨C#, JS⟩, ⟨Python, JS⟩ and ⟨JS, Java⟩, where JS refers to JavaScript.

All experiments were performed on a machine with an Intel i7 processor having a processing speed of 2.10 GHz, 16 GB of memory, and running on Ubuntu 16.04 LTS operating system.

D. Evaluation Strategies

We need to manually validate analogical library recommendations to determine which are true analogical libraries and to evaluate the performance of compared techniques. A group of eight human evaluators manually validated those recommendations. To avoid bias we did not disclose the technique names to the evaluators. We divided the evaluators into two different groups of equal sizes. To evaluate the performance of a technique, each group was given the Top-5 recommendations made for 450 libraries by that technique. In total, the two groups evaluated the recommendations made for 900 libraries. Once we received the evaluation results, we alternate the libraries between the groups for another round of evaluation. Thus, the analogical library recommendations for a library were validated by two different evaluators. In case of conflicts, the evaluators talked to each other to resolve the issue. Otherwise, we removed the library from our analysis. However, the evaluators were able to resolve all the conflicts.

E. Evaluation Metric

We use the precision to measure the performance of our compared techniques. Our selection of the evaluation metric is based on the fact that a number of prior studies used the metric [6], [18], [19]. For each test-case library, we obtained the Top-k recommendations for each compared technique, assuming that the technique recommends at least one library. Next, we determine how many of those recommendations are true analogical libraries. Thus, the Precision@K can be defined as the ratio of true analogical libraries over the Top-k recommended libraries. Next, we take the average of the results for the set of test cases as follows:

\[
\sum_{i=1}^{n} \frac{\text{Precision}_i@k}{n}
\]

IV. EXPERIMENTAL RESULTS

This section presents our evaluation results and answers to our research questions.

A. Research Questions

1) RQ1: How effective is XLibRec in recommending analogical libraries for Chen et al.’s dataset (Dataset-A’)? To get the answer for RQ1 we observed the comparison of both of the techniques’ recommendations in the Top-1, Top-3 and Top-5 positions. As we didn’t have any prior knowledge about our collected datasets, we only considered the precision of the analogical library recommendations based on their recommended positions. From Table [I] we can see that, for different combinations of languages such as ⟨Java, Python⟩, ⟨Java, C#⟩, ⟨C#, JavaScript⟩, ⟨Python, JavaScript⟩ and ⟨JavaScript, Java⟩, for the Top-1 position, Chen et al.’s SimilarTech got around
TABLE II: Cross-Language analogical library recommendation comparison with Dataset-A’

<table>
<thead>
<tr>
<th>(Source, Dest.)</th>
<th>SimilarTech (Precision)</th>
<th>XLibRec (Precision)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-1</td>
<td>Top-3</td>
</tr>
<tr>
<td>(Java, Python)</td>
<td>0.62</td>
<td>0.70</td>
</tr>
<tr>
<td>(Java, C#)</td>
<td>0.59</td>
<td>0.74</td>
</tr>
<tr>
<td>(C#, JS)</td>
<td>0.58</td>
<td>0.68</td>
</tr>
<tr>
<td>(Python, JS)</td>
<td>0.55</td>
<td>0.66</td>
</tr>
<tr>
<td>(JS, Java)</td>
<td>0.64</td>
<td>0.72</td>
</tr>
</tbody>
</table>

TABLE III: Cross-Language analogical library recommendation comparison with Dataset-B

<table>
<thead>
<tr>
<th>(Source, Dest.)</th>
<th>SimilarTech (Precision)</th>
<th>XLibRec (Precision)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-1</td>
<td>Top-3</td>
</tr>
<tr>
<td>(Java, Python)</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td>(Java, C#)</td>
<td>0.51</td>
<td>0.54</td>
</tr>
<tr>
<td>(C#, JS)</td>
<td>0.44</td>
<td>0.49</td>
</tr>
<tr>
<td>(Python, JS)</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>(JS, Java)</td>
<td>0.49</td>
<td>0.32</td>
</tr>
</tbody>
</table>

60% precision whereas our proposed technique XLibRec recommends with around 74% or more accuracy on average in terms of precision. As we considered library usage context which can capture the functionality and scope of a library more precisely than other sources, our technique performs better than the baseline method (i.e. SimilarTech). For Top-3 recommendation positions, Chen’s technique obtained around 68% on average whereas XLibRec has observed around 77% precision. The reason we investigated for getting better results is the same as we discussed for the Top-1 recommendation position. Moreover, for Top-5 positions, Chen et al. obtained around 65% precision and our technique experienced around 70% or more precision on average which is 5% more than the baseline method.

2) RQ2: How effective is XLibRec in recommending analogical libraries for Dataset-B? To answer RQ2, we executed XLibRec and SimilarTech on Dataset-B’, validated the recommendations and recorded the results in Table III. From the table, we see that our technique recommends analogical libraries for the top-1, top-3 and top-5 positions with an average precision of 72% or more, which is around 30% better than the baseline technique SimilarTech. We investigated the reason behind our better performance. SimilarTech is fully dependent on the association of tags and the similarity between two library tags. The libraries which have fewer posts in SO, have less possibility of having similar tags co-occurring with their analogical ones. On the other hand, our technique focuses on library usage information and the functional description of libraries. As a consequence, XLibRec successfully recommended analogical libraries with higher precision than the baseline method.

3) RQ3: How effective is XLibRec in recommending analogical libraries for Dataset-C? We recorded our investigation results in Table IV for answering RQ3. SimilarTech requires that the library name should be used as tags in the SO questions. Thus, it is not able to recommend without library tag information. However, Dataset-C’ consists of those libraries that are not used as tags in SO. Thus, we decided not to use the baseline method to answer this question. So, we only execute our proposed technique on this dataset. Our evaluation shows that for Dataset-C’ our technique can recommend analogical libraries among four different languages with around 65% precision which is around 6-10% lower than RQ1 and RQ2. This is because we could not find the discussion of some libraries in SO posts and were unable to collect the LUI. Thus, we needed to solely depend on the library description for those cases. As a consequence, our technique recommended a couple of false-positive analogical libraries which hinders the overall average performance of our technique a bit, but still it could recommend analogical libraries with more than 65% precision, which is promising.

The above results show that XLibRec performs better than SimilarTech. We hypothesize that the use of library usage information and different sources of information help XLibRec to better identify analogical libraries. To provide further insights on our hypothesis, we conduct additional experiments and present the results in this section.

B. Sensitivity Analysis: Impact of different information sources

We considered each of the information sources (SO question titles, answer bodies, and library descriptions) as individual models and recorded their performances in recommending cross-language analogical libraries. We also considered a combination of these three sources as individual models to see their performance. In addition to these we considered SO tags as an additional source of information, build a model considering the information and also considered its combination with previous models to see whether they could outperform other sources. In short, we proceeded with further investigation with the following variations of information sources and their models to clarify the independent impact of each of the sources in recommending cross-language similar libraries and their probable combinations:

- S1: LUI from SO answer body
- S2: LUI from SO question title
- S3: SO post tags
- S4: LUI from Library description
- S5: S1 + S2
- S6: S1 + S2 + S3
- S7: S1 + S4
- S8: S1 + S2 + S4
- S9: S1 + S3 + S4
Developers often use more than one line to describe the functionalities of a library. At the same time, some of their descriptions only partially describe the functionalities of a library in SO answer bodies. The same situation goes for SO question titles. This depicts that the description of a library provided by the developers captures the detail description of functionalities of that library. This result is promising, but still the precision would remain lower than 70% which motivated us to see whether the combination of information sources would increase the recommendation accuracy.

To experience the performance of the combination of information sources we first combined library usage descriptions mined from SO answer bodies with library usage information mined from SO post titles. For this combination, we observed around 57% accuracy in recommending cross-language analogical libraries which is similar to the model considering question titles. This is because some of those lines contain irrelevant knowledge and library usages along with other information when answering a question in SO. To determine how many lines around a library name in a SO answer body would actually contain the library usage information (LUI) we performed an additional experiment with the libraries we selected from our three datasets (Dataset-$A'$, Dataset-$B'$, Dataset-$C'$). Figure 4 shows the precision of our technique when using different number of lines from SO answer bodies to collect LUI. Here, the term OneLine refers to the line on which the library name appears in the SO answer body; the term TwoLines indicates the OneLine plus the next line in SO answer body, the term ThreeLines refers to the TwoLines plus next line and this continues for others. Finally, the term Paragraph refers to paragraph section that contains the library name. From the figure we can see that our technique performs the same in recommending cross-language analogical libraries when using Paragraph and ThreeLines for collecting LUI. This is because in most of the cases the length of the paragraph is three lines. The performance of the technique does not necessarily improve when considering more lines to collect the LUI. This is because some of those lines contain irrelevant information and this negatively affects the performance of the technique. The technique performs the best when we collect LUI considering two lines, can recommend cross-language analogical libraries with an average precision of 60% and above. That is why we considered TwoLines to collect LUI.

D. Overlapping of recommendations

In this section, we are interested in learning how XLibRec and SimilarTech complement each other. For this study, we consider the Top-1 recommendation. We also manually analyzed the recommendation made by XLibRec and SimilarTech for Dataset-$A'$ and Dataset-$B'$.

Figure 5 shows the overlapping of recommendations for the Top-1 position between XLibRec and SimilarTech techniques. From the figure we can see that for the Top-1 recommendation and for 67.8% of test cases both of the techniques

\[ S_{10'}: S_2 + S_3 + S_4 \]

\[ S_{11'}: S_1 + S_2 + S_3 + S_4 \]

We performed this experiment again on the randomly selected libraries from all three datasets and recorded in the Table. From the table we could see that, if XLibRec depended only on mining library usage descriptions using word2vec from SO answer body it could recommend cross-language analogical libraries with around 45% precision, which is promising but not up to the mark. This is because developers use different words to state the same functionality of a library. At the same time, some of their descriptions only partially describe the functionalities of a library in SO answer bodies. The same situation goes for SO question titles which results lower than 55% accuracy in recommending cross-language analogical libraries. Next, we considered the similarity of tags co-occurrence in SO posts in between libraries as our third model. For this model we got very low accuracy, lower than 32%. Finally, if we considered only library descriptions, we would have a probability of getting an average precision of 55% in recommending cross-language analogical libraries which is similar to the model considering question titles. This depicts that the description of a library provided by the developers captures the detail description of functionalities of that library. This result is promising, but still the precision would remain lower than 70% which motivated us to see whether the combination of information sources would increase the recommendation accuracy.

To experience the performance of the combination of information sources we first combined library usage descriptions mined from SO answer bodies with library usage information mined from SO post titles. For this combination, we observed around 57% accuracy in recommending cross-language analogical libraries which is similar to the model considering question titles. This is because some of those lines contain irrelevant knowledge and library usages along with other information when answering a question in SO. To determine how many lines around a library name in a SO answer body would actually contain the library usage information (LUI) we performed an additional experiment with the libraries we selected from our three datasets (Dataset-$A'$, Dataset-$B'$, Dataset-$C'$). Figure 4 shows the precision of our technique when using different number of lines from SO answer bodies to collect LUI. Here, the term OneLine refers to the line on which the library name appears in the SO answer body; the term TwoLines indicates the OneLine plus the next line in SO answer body, the term ThreeLines refers to the TwoLines plus next line and this continues for others. Finally, the term Paragraph refers to paragraph section that contains the library name. From the figure we can see that our technique performs the same in recommending cross-language analogical libraries when using Paragraph and ThreeLines for collecting LUI. This is because in most of the cases the length of the paragraph is three lines. The performance of the technique does not necessarily improve when considering more lines to collect the LUI. This is because some of those lines contain irrelevant information and this negatively affects the performance of the technique. The technique performs the best when we collect LUI considering two lines, can recommend cross-language analogical libraries with an average precision of 60% and above. That is why we considered TwoLines to collect LUI.

C. Sensitivity Analysis: Impact of using different number of lines

Determining how many lines around a library can accurately describe the LUI of that library is a non-trivial task. Developers often use more than one line to describe the library usages along with other information when answering a question in SO. To determine how many lines around a library name in a SO answer body would actually contain the library usage information (LUI) we performed an additional experiment with the libraries we selected from our three datasets (Dataset-$A'$, Dataset-$B'$, Dataset-$C'$). Figure 4 shows the overlapping of recommendations for the Top-1 position between XLibRec and SimilarTech techniques. From the figure we can see that for the Top-1 recommendation and for 67.8% of test cases both of the techniques

\[ S_{10'}: S_2 + S_3 + S_4 \]

\[ S_{11'}: S_1 + S_2 + S_3 + S_4 \]

We performed this experiment again on the randomly selected libraries from all three datasets and recorded in the Table. From the table we could see that, if XLibRec depended only on mining library usage descriptions using word2vec from SO answer body it could recommend cross-language analogical libraries with around 45% precision, which is promising but not up to the mark. This is because developers use different words to state the same functionality of a library. At the same time, some of their descriptions only partially describe the functionalities of a library in SO answer bodies. The same situation goes for SO question titles which results lower than 55% accuracy in recommending cross-language analogical libraries. Next, we considered the similarity of tags co-occurrence in SO posts in between libraries as our third model. For this model we got very low accuracy, lower than 32%. Finally, if we considered only library descriptions, we would have a probability of getting an average precision of 55% in recommending cross-language analogical libraries which is similar to the model considering question titles. This depicts that the description of a library provided by the developers captures the detail description of functionalities of that library. This result is promising, but still the precision would remain lower than 70% which motivated us to see whether the combination of information sources would increase the recommendation accuracy.

To experience the performance of the combination of information sources we first combined library usage descriptions mined from SO answer bodies with library usage information mined from SO post titles. For this combination, we observed around 57% accuracy in recommending cross-language analogical libraries which is similar to the model considering question titles. This is because some of those lines contain irrelevant knowledge and library usages along with other information when answering a question in SO. To determine how many lines around a library name in a SO answer body would actually contain the library usage information (LUI) we performed an additional experiment with the libraries we selected from our three datasets (Dataset-$A'$, Dataset-$B'$, Dataset-$C'$). Figure 4 shows the precision of our technique when using different number of lines from SO answer bodies to collect LUI. Here, the term OneLine refers to the line on which the library name appears in the SO answer body; the term TwoLines indicates the OneLine plus the next line in SO answer body, the term ThreeLines refers to the TwoLines plus next line and this continues for others. Finally, the term Paragraph refers to paragraph section that contains the library name. From the figure we can see that our technique performs the same in recommending cross-language analogical libraries when using Paragraph and ThreeLines for collecting LUI. This is because in most of the cases the length of the paragraph is three lines. The performance of the technique does not necessarily improve when considering more lines to collect the LUI. This is because some of those lines contain irrelevant information and this negatively affects the performance of the technique. The technique performs the best when we collect LUI considering two lines, can recommend cross-language analogical libraries with an average precision of 60% and above. That is why we considered TwoLines to collect LUI.

D. Overlapping of recommendations

In this section, we are interested in learning how XLibRec and SimilarTech complement each other. For this study, we consider the Top-1 recommendation. We also manually analyzed the recommendation made by XLibRec and SimilarTech for Dataset-$A'$ and Dataset-$B'$.

Figure 5 shows the overlapping of recommendations for the Top-1 position between XLibRec and SimilarTech techniques. From the figure we can see that for the Top-1 recommendation and for 67.8% of test cases both of the techniques
TABLE V: Impact of individual sources in detecting cross-language similar libraries

<table>
<thead>
<tr>
<th>Model</th>
<th>Java → Python</th>
<th>Java → C#</th>
<th>C# → JavaScript</th>
<th>Python → JavaScript</th>
<th>JavaScript → Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-1</td>
<td>Top-3</td>
<td>Top-5</td>
<td>Top-1</td>
<td>Top-3</td>
</tr>
<tr>
<td>S1</td>
<td>0.52</td>
<td>0.47</td>
<td>0.5</td>
<td>0.5</td>
<td>0.54</td>
</tr>
<tr>
<td>S2</td>
<td>0.54</td>
<td>0.57</td>
<td>0.5</td>
<td>0.5</td>
<td>0.52</td>
</tr>
<tr>
<td>S3</td>
<td>0.31</td>
<td>0.36</td>
<td>0.36</td>
<td>0.29</td>
<td>0.34</td>
</tr>
<tr>
<td>S4</td>
<td>0.47</td>
<td>0.58</td>
<td>0.49</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>S5</td>
<td>0.33</td>
<td>0.57</td>
<td>0.56</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>S6</td>
<td>0.44</td>
<td>0.43</td>
<td>0.43</td>
<td>0.47</td>
<td>0.44</td>
</tr>
<tr>
<td>S7</td>
<td>0.52</td>
<td>0.54</td>
<td>0.57</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>S8</td>
<td>0.72</td>
<td>0.76</td>
<td>0.75</td>
<td>0.74</td>
<td>0.77</td>
</tr>
<tr>
<td>S9</td>
<td>0.52</td>
<td>0.55</td>
<td>0.55</td>
<td>0.5</td>
<td>0.57</td>
</tr>
<tr>
<td>S10</td>
<td>0.51</td>
<td>0.35</td>
<td>0.35</td>
<td>0.53</td>
<td>0.58</td>
</tr>
<tr>
<td>S11</td>
<td>0.48</td>
<td>0.53</td>
<td>0.46</td>
<td>0.51</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Fig. 5: Venn diagram showing the overlapping of recommendation between XLibRec and SimilarTech

recommended the same analogical library. While XLibRec was successful to recommend the correct analogical library for 20.9% of test cases, SimilarTech was not successful for those cases. On the contrary, we observed 11.3% of test cases where SimilarTech was successful in recommending correct cross-language analogical libraries in the Top-1 position that were not detected by XLibRec. There are two implications of our findings. First, our technique is able to make the correct recommendation for more cases compared to SimilarTech, showing the importance of our technique. Furthermore, we also see an opportunity in combining recommendations of two techniques (i.e., XLibRec, SimilarTech) to improve the overall performance. Future research can focus on this direction.

E. Comparison with Google Search

This section describes a comparison of our proposed technique with Google Search. Nowadays, Google Search is a standard approach for finding any information [20]. Developers can search for analogical libraries through Google to solve their problems. Google Search returns pages of information related to the search query where a developer needs to analyze every page to get the analogical library information. Thus, we considered Google Search as our another baseline technique and tried to see how Google Search performs in recommending cross-language analogical libraries.

To perform the Google Search we configured search queries with the two following patterns:

\[ a. [\text{library\_name}] [\text{keywords}] [\text{language\_name}] \]  
\[ b. [\text{language\_name}] [\text{keywords}] [\text{library\_name}] \]  

In these patterns, the following information is required to provide:

- \text{library\_name} expects the name of the library for which the developer is looking for the analogical libraries.
- \text{language\_name} expects the name of the programming language for which the developer is looking for the analogical library.
- \text{keywords} is a list of words that can be used in the query to conduct the search. We observed three patterns of keywords that could be used as queries to find analogous libraries. These are: i. alternate library in, ii. similar library in, iii. equivalent library in

Following these patterns we performed 200 library queries, 50 for each of the programming languages, in a random manner. In other words, we randomly collected 200 libraries from Dataset-\(A'\), Dataset-\(B'\) and Dataset-\(C'\) for \text{library\_name}, and randomly selected the target \text{language\_name}. We used both of the patterns equally to perform the queries. For each search query, we randomly selected \text{library\_name} and \text{language\_name}. Next, we used all three variations in \text{keywords} to perform the search and recorded the returned results. Table VI records the Google Search performance with the variations in \text{keywords}.

TABLE VI: Google Search Performance in Recommending Analogical Library in Top-k ranked positions

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Top-1</th>
<th>Top-5</th>
<th>Top-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Library in</td>
<td>0.14</td>
<td>0.32</td>
<td>0.48</td>
</tr>
<tr>
<td>Similar Library in</td>
<td>0.20</td>
<td>0.37</td>
<td>0.57</td>
</tr>
<tr>
<td>Equivalent Library in</td>
<td>0.07</td>
<td>0.22</td>
<td>0.41</td>
</tr>
</tbody>
</table>

For Google Search we tried to find out how many search queries perfectly returned a solution page in Top-1, Top-5 and Top-10 ranked positions. The reason behind selecting Top-10 is to cover all the search results those appear in the first page of a Google search. If Google Search returned a page that consists of an accurate analogical library of the given search query we considered that query a successful one. Next, the ranked position of that page in Google Search defines the recommendation position at which that query became successful. From table VI we can see that, for the Top-1 recommendation, Google Search successfully recommended
analogue libraries for less than 20% of cases. Most of these search results refer to SO posts where a developer was asking for a cross-language analogue library. A major drawback of the approach is that developers need to manually visit the whole answer of the question to find out the appropriate library they are looking for. Such an approach is a time-consuming and costly operation. Furthermore, there is no guarantee that the developer can find the analogue library they are looking for. We developed our techniques considering the above challenges. Our proposed technique mined the LUI of each of the libraries from Stack Overflow posts and library description, and finally came up with a better recommendation of analogue libraries. We checked the performance of Google Search for the Top-5 and Top-10 recommendations. Results from our evaluation revealed the poor performance of Google Search in finding analogue libraries. The query success rate remains lower than 50% on average which indicates the importance of our research and tool support to help developers in finding cross-language software applications.

V. Threats to Validity

This section discusses the threats to the validity of this study. We evaluate our proposed technique using a collection of libraries. One might argue that the results we obtain may not generalize to other libraries and programming languages. However, all our selected libraries are actively used by open source software systems, provide a wide range of functionalities (e.g., machine learning, natural language processing, visualization, and so on), and are collected from four popular programming languages. Thus our results should be relevant to a broad range of cases.

The performance of our proposed technique can be affected by the ability to determine whether a recommended library is a true analogue library or not. To mitigate this issue, each recommendation was validated separately by two different evaluators. In case of any ambiguity, the evaluators talked to each other to resolve the conflict. Otherwise, we removed the library from our study. However, we did not encounter such cases.

We re-implemented SimilarTech as the implementation of the technique is not publicly available. Although we cannot guarantee that our implementation does not contain any errors, we spent considerable time implementing and testing the technique to avoid introducing any errors.

VI. Related Work

Researchers have been conducting research on automatic recommendation systems in software engineering for a long period of time. Moreover, researchers have been using different approaches to detect similar libraries both cross-language and within a single programming language.

A. Analogue Library Recommendation

Research on recommending analogue library is going on for a long time. The applications of this research take place at the time of developing a similar software application in a different language, extending an application with a new functionality, or developing a different software application with some of the same functionality. In analogue library recommendation, researchers try to detect similar libraries by exploiting functional similarity between two libraries [21], library usage patterns [2] or exploring a third-party information source such as Stack Overflow [5-7] to mine relation between libraries. They try to extract analogue third-party libraries across different programming languages by incorporating relational, categorical and semantic information from Stack Overflow. Among all these works, Chen et al.’s [6] work, SimilarTech is directly related to our technique. SimilarTech can detect similar libraries across different programming languages without any predefined knowledge or functionally similar code blocks, but it totally depends on Stack Overflow user experience and question tags which could not explore all the relationships among the libraries available for development. Moreover, their work totally depends on Stack Overflow library tags and fails to cover all the available libraries (i.e. libraries that are not used as tags). Our technique overcomes these limitations and can recommend libraries not only available in Stack Overflow but also the libraries available across different package managers of different programming languages. We did not consider tags co-occurrence information as in the long run they fail to establish relations in between analogue libraries.

B. Analogue Software Application Recommendation

Our work is not directly connected with detecting and categorizing analogue software applications [22-25] in a software repository, but it could be applicable as many of these use library similar information in the source code to detect similar software applications. For example, McMillan et al. exploited library name and API Call similarity between two applications in Java [22] and Nafti et al. [26] explored this information for detecting and categorizing cross-language similar software applications in a software repository. As our work is related to cross-language analogous library detection and recommendation we can extend our technique in detecting cross-language analogous software applications in the near future.

C. Analogue API Recommendation

Researchers are working on recommending analogue APIs to help developers at the time of developing a software application using one or more programming languages. At the same time, research is focusing on helping developers with automatic code migration by migrating the API of one library to the API of another library. Researchers use different ways to mine API similarity such as code mapping [27], function mapping [21], mining API usage patterns [28], functional description mapping [29] and so on. Teytoun et al. [21] infer likely API mappings between similar libraries by examining already-ported code, i.e., changes made to port an application from using one library’s APIs to another library’s. Santhiar et al. [30] mine similar units tests for migrating math APIs and Chen.
et al. [31], [32] mined API usage patterns from source code of different software applications. All these works are hard to adapt in cross-language software development environment as it is hard to get pre-approved library change information among two different software applications developed in two different programming languages. In addition, some works do not require predefined code change information. For example, Pandita et al. [33] use text mining to identify likely API mappings based on the textual similarity of API names and documents, and Lu et al. [34] and Gu et al. [35] infer similar APIs across different programming languages from API documents, mining API usage patterns [36], [37] and recommending API usage examples [38]. API mapping is possible even when there doesn’t exist any direct relation between APIs used in different programming languages other than some functional description similarity [39]. As our technique is based on mining the library usage information from SO posts and library descriptions, we can extend this technique to mine API descriptions too. For now, this research direction is out of the scope of this paper.

D. Word Embedding System in Software Engineering

Our technique is directly based on the success of recent flourishing of word embedding techniques such as Word2Vec [9], Paragraph2Vec [40] and so on. Many researchers adapted these models to solve problems in software engineering such as information retrieval from documentations [41], API-level code migration [42], API usage pattern embedding [31], [43] and so on. Apart from these works, in our technique, we applied Word2Vec to vectorize the LUI collected from SO posts and library descriptions which we finally extended to recommend cross-language analogical libraries.

E. Library Migration

Our work could be considered with library migration (i.e. upgrading the libraries used in a source code from its previous version to a newer one or replaced with another upgraded library). A good number of researchers have already worked on similar library detection and tried to get a stable solution that can be extended for automatic program or library migration [19], [44]–[50]. Some of them tried to mine library information from source code [45], [46], [49]. Some of others tried to mine library usage patterns using that information [44]. Most of these works are supported for a single language platform rather than in cross-language software development. Some techniques support cross-language in a different manner. For example, the use of a single tool in different programming languages such as Lucene can be used with almost the same features in Java and C# [51]. In this case, two functionally similar tools are required to migrate libraries from one to another which is separated from our case. We tried to cover not only functionally similar libraries for different versions of a tool but also the libraries which are not present in a tool but functionally similar in different manners. Our technique could be extended to use as a library migration tool for cross-language developed applications as well. But as this is out of the present scope of this paper, we didn’t show any comparison or analysis to any of the models related to library migration at this point.

F. Task Navigation

Research on task navigation refers to the process of information retrieval and locating important information in software application documentation or other related documents. This helps developers to steer through the important information related to library and APIs in different documentation at the time of developing and maintaining software applications [10], [52]–[63]. Among them, Treude et al. [10] exploited Stack Overflow posts to develop API documentation where they extracted task performed by an API. They also tried to mine some patterns to easily navigate in software engineering documentations. In another work, Nadi et. al [63] mined some sentences which helped developers to navigate to a SO post based on their task and context. In contrary to these works, we tried to mine some text patterns that developers usually use when recommending a library in Stack Overflow. This helped us to filter library related posts out of all posts. By relating a library with LUI extracted from SO answer bodies and question titles, we generated an experience-based functional description of that library which is not directly related to the basic goal of task navigation models.

VII. Conclusion

In this paper, we present a technique, called XLibRec, to recommend analogical libraries across different programming languages. We collect library usage information by mining online posts and library descriptions. We leverage the word embedding method to generate word vectors from library usage information and apply Cosine similarity to calculate a similarity score between a pair of libraries. Such scores can help us to recommend analogical libraries for a given library. Our evaluation of XLibRec with a state-of-the-art technique, SimilarTech, shows that XLibRec can outperform SimilarTech by 8-45% higher precision. Our technique can make recommendations even when the libraries are not used as tags in Stack Overflow questions. Thus, the technique can make recommendations for a large number of libraries.

In the future, we plan to identify additional aspects of libraries by mining developer opinions of libraries posted on the web. This can further help developers to select one library over another. We also plan to extend the technique to find analogical APIs. Questions regarding analogical APIs are also common in Stack Overflow. Information regarding analogical libraries and APIs can also assist developers to migrate from one language to another, and in particular when migrating to a language with which they are not familiar.

ACKNOWLEDGEMENT

This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants, and by an NSERC Collaborative Research and Training Experience (CREATE) grant.


