Tell Them Apart: Distilling Technology Differences from Crowd-Scale Comparison Discussions

Yi Huang
Research School of Computer Science, Australian National University
Australia
u6039034@anu.edu.au

Chunyang Chen*
Faculty of Information Technology, Monash University
Australia
chunyang.chen@monash.edu

Zhenchang Xing
Research School of Computer Science, Australian National University
Australia
zhenchang.xing@anu.edu.au

Tian Lin, Yang Liu
School of Computer Science and Engineering, Nanyang Technological University
Singapore
{lint0011,yangliu}@ntu.edu.sg

ABSTRACT

Developers can use different technologies for many software development tasks in their work. However, when faced with several technologies with comparable functionalities, it is not easy for developers to select the most appropriate one, as comparisons among technologies are time-consuming by trial and error. Instead, developers can resort to expert articles, read official documents or ask questions in Q&A sites for technology comparison, but it is opportunistic to get a comprehensive comparison as online information is often fragmented or contradictory. To overcome these limitations, we propose the *diffTech* system that exploits the crowd-sourced discussions from Stack Overflow, and assists technology comparison with an informative summary of different comparison aspects. We first build a large database of comparable technologies in software engineering by mining tags in Stack Overflow, and then locate comparative sentences about comparable technologies with natural language processing methods. We further mine prominent comparison aspects by clustering similar comparative sentences and representing each cluster with its keywords. The evaluation demonstrates both the accuracy and usefulness of our model and we implement our approach into a practical website for public use.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Software and its engineering** → **Software libraries and repositories**;

KEYWORDS

differencing similar technology, Stack Overflow, NLP

ACM Reference Format:
Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell Them Apart: Distilling Technology Differences from Crowd-Scale Comparison Discussions. In *Proceedings of the 2018 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18), September 3–7, 2018, Montpellier, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3238147.3238208

*Co-first and corresponding author.

ASE '18, September 3–7, 2018, Montpellier, France

ACM ISBN 978-1-4503-5937-5/18/09…$15.00
https://doi.org/10.1145/3238147.3238208

Figure 1: A comparative sentence in a post (#1008671) that is not explicitly for technology comparison

1 INTRODUCTION

A diverse set of technologies (e.g, algorithms, programming languages, platforms, libraries/frameworks, concepts for software engineering) [12, 15] is available for use by developers and that set continues growing. By adopting suitable technologies, it will significantly accelerate the software development process and also enhance the software quality. But when developers are looking for proper technologies for their tasks, they are likely to find several comparable candidates. For example, they will find *bubble sort* and *quick sort* algorithms for sorting, *nltk* and *opennlp* libraries for NLP, *Eclipse* and *Intellij* for developing Java applications.

Faced with so many candidates, developers are expected to have a good understanding of different technologies in order to make a proper choice for their work. However, even for experienced developers, it can be difficult to keep pace with the rapid evolution of technologies. Developers can try each of the candidates in their work for the comparison. But such trial-and-error assessment is time-consuming and labor extensive. Instead, we find that the perceptions of developers about comparable technologies and the choices they make about which technology to use are very likely to be influenced by how other developers *see* and *evaluate* the technologies. So developers often turn to the two information sources on the Web [8] to learn more about comparable technologies.

First, they read experts' articles about technology comparison like *"Intellij vs. Eclipse: Why IDEA is Better"*. Second, developers can seek answers on Q&A websites such as Stack Overflow or Quora (e.g., *"Apache OpenNLP vs NLTK"*). These expert articles and community answers are indexable by search engines, thus enabling developers to find answers to their technology comparison inquiries.

However, there are two limitations with expert articles and community answers.

- *Fragmented view:* An expert article or community answer usually focuses on a specific aspect of some comparable technologies, and developers have to aggregate the fragmented information into a complete comparison in different aspects. For example, to compare *mysql* and *postgresql*, one article [2] contrasts their speed, while another [3] compares their reliability. Only after reading both articles, developers can have a relatively comprehensive overview of these two comparable technologies.
- *Diverse opinions:* One expert article or community answer is based on the author's knowledge and experience. However, the knowledge and experience of developers vary greatly. For example, one developer may prefer *Eclipse* over *Intellij* because *Eclipse* fits his project setting better. But that setting may not be extensible to other developers. At the same time, some developers may prefer *Intellij* over *Eclipse* for other reasons. Such contradictory preferences among different opinions may confuse developers.

The above two limitations create a high barrier for developers to effectively gather useful information about technology differences on the Web in order to tell apart comparable technologies. Although developers may manually aggregate relevant information by searching and reading many web pages, that would be very opportunistic and time consuming. To overcome the above limitations, we present the *diffTech* system that automatically distills and aggregates fragmented and trustworthy technology comparison information from the crowd-scale Q&A discussions in Stack Overflow, and assists technology comparison with an informative summary of different aspects of comparison information.

Our system is motivated by the fact that a wide range of technologies have been discussed by millions of users in Stack Overflow [14], and users often express their preferences toward a technology and compare one technology with the others in the discussions. Apart from posts explicitly about the comparison of some technologies, many comparative sentences **hide** in posts that are implicitly about technology comparison. Fig.1 shows such an example: the answer "accidentally" compares the security of *POST* and *GET*, while the question "How secure is a HTTP post?" does not explicit ask for this comparison. Inspired by such phenomenon, we then propose our system to mine and aggregate the comparative sentences in Stack Overflow discussions.

As shown in Fig. 2, we consider Stack Overflow tags as a collection of technology terms and first find comparable technologies by analyzing tag embeddings and categories. And then, our system distills and clusters comparative sentences from Q&A discussions, which highly likely contains detailed comparisons between some comparable technologies, and sometimes even explains why users like or dislike a particular technology. Finally, we use word mover distance [28] and community detection [21] to cluster comparative sentences into prominent aspects by which users compare the two technologies and present the mined clusters of comparative sentences for user inspection.

As there is no ground truth for technology comparison, we manually validate the performance of each step of our approach. The experiment results confirm the the accuracy of comparable technology identification (90.7%), and distilling comparative sentences (83.7%) from Q&A discussions. By manually building the ground truth, we show that our clustering method (word mover distance and community detection) for comparative sentences significantly outperforms the two baselines (TF-IDF with K-means and Doc2vec with K-means). Finally, we further demonstrate the usefulness of our system for answering questions of technology comparison in Stack Overflow. The result show that our system can cover the semantics of 72% comparative sentences in five randomly selected technology comparison questions, and also include some unique comparisons from other aspects which are not discussed in original answers.

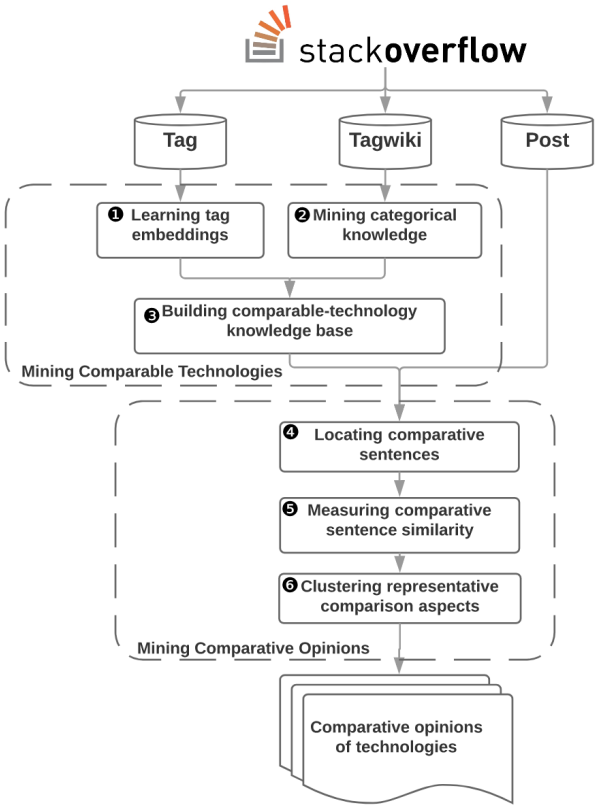
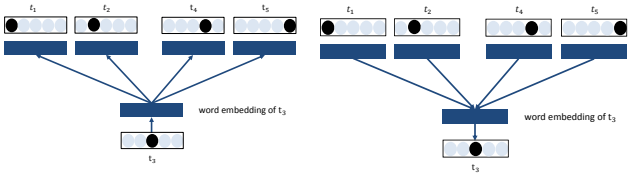


Figure 2: The overview of our approach

Our contributions in this work are four-fold:

- This is the first work to systematically identify comparable software-engineering technologies and distill crowd-scale comparative sentences for these technologies.
- Our method automatically distills and aggregates crowd opinions into different comparison aspects so that developers can understand technology comparison more easily.
- Our experiments demonstrate the effectiveness of our method by checking the accuracy and usefulness of each step of our approach.
- We implement our results into a practical tool and make it public to the community. Developers can benefit from the technology comparison knowledge in our website[1].

[1] https://difftech.herokuapp.com/



(a) Continuous skip-gram model (b) Continuous bag-of-words model

Figure 3: The architecture of the two word embeddings models. The continuous skip-gram model predicts surrounding words given the central word, and the CBOW model predicts the central word based on the context words. Note the differences in arrow direction between the two models.

2 MINING SIMILAR TECHNOLOGY

Studies [9, 11, 44] show that Stack Overflow tags identify computer programming technologies that questions and answers revolve around. They cover a wide range of technologies, from algorithms (e.g., *binary search, merge sort*), programming languages (e.g., *python, java*), libraries and frameworks (e.g., *tensorflow, django*), and development tools (e.g., *vim, git*). In this work, we regard Stack Overflow tags as a collection of technologies that developers would like to compare. We leverage word embedding techniques to infer semantically related tags, and develop natural language methods to analyze each tag's TagWiki to determine the corresponding technology's category (e.g., algorithm, library, IDE). Finally, we build a knowledge base of comparable technologies by filtering the same-category, semantically-related tags.

2.1 Learning Tag Embeddings

Word embeddings are dense low-dimensional vector representations of words that are built on the assumption that words with similar meanings tend to be present in similar context. Studies [11, 35] show that word embeddings are able to capture rich semantic and syntactic properties of words for measuring word similarity. In our approach, given a corpus of tag sentences, we use word embedding methods to learn the word representation of each tag using the surrounding context of the tag in the corpus of tag sentences.

There are two kinds of widely-used word embedding methods [35], the continuous skip-gram model [36] and the continuous bag-of-words (CBOW) model. As illustrated in Fig. 3, the objective of the continuous skip-gram model is to learn the word representation of each word that is good at predicting the co-occurring words in the same sentence (Fig. 3(a)), while the CBOW is the opposite, that is, predicting the center word by the context words (Fig. 3(b)). Note that word order within the context window is not important for learning word embeddings.

Specifically, given a sequence of training text stream $t_1, t_2, ..., t_k$, the objective of the continuous skip-gram model is to maximize the following average log probability:

$$L = \frac{1}{K}\sum_{k=1}^{K}\sum_{-N\le j\le N, j\neq 0} \log p(t_{k+j}|t_k) \quad (1)$$

Tag Wiki:	Matplotlib	**is**	a	plotting	**library**	for	Python
Part of Speech:	NNP	VBZ	DT	JJ	**NN**	IN	NNP

Figure 4: POS tagging of the definition sentence of the tag *Matplotlib*

while the objective of the CBOW model is:

$$L = \frac{1}{K}\sum_{k=1}^{K} \log p(t_k|(t_{k-N}, t_{k-N+1}, ..., t_{k+N})) \quad (2)$$

where t_k is the central word, t_{k+j} is its surrounding word with the distance j, and N indicates the window size. In our application of the word embedding, a tag sentence is a training text stream, and each tag is a word. As tag sentence is short (has at most 5 tags), we set N as 5 in our approach so that the context of one tag is all other tags in the current sentences. That is, the context window contains all other tags as the surrounding words for a given tag. Therefore, tag order does not matter in this work for learning tag embeddings.

To determine which word-embedding model performs better in our comparable technology reasoning task , we carry out a comparison experiment, and the details are discussed in Section 5.1.3.

2.2 Mining Categorical Knowledge

In Stack Overflow, tags can be of different categories, such as programming language, library, framework, tool, API, algorithm, etc. To determine the category of a tag, we resort to the tag definition in the TagWiki of the tag. The TagWiki of a tag is collaboratively edited by the Stack Overflow community. Although there are no strict formatting rules in Stack Overflow, the TagWiki description usually starts with a short sentence to define the tag. For example, the tagWiki of the tag *Matplotlib* starts with the sentence "Matplotlib is a plotting library for Python". Typically, the first noun just after the *be* verb defines the category of the tag. For example, from the tag definition of *Matplotlib*, we can learn that the category of *Matplotlib* is *library*.

Based on the above observation of tag definitions, we use the NLP methods [11, 27] to extract such noun from the tag definition sentence as the category of a tag. Given the tagWiki of a tag in Stack Overflow, we extract the first sentence of the TagWiki description, and clean up the sentence by removing hyperlinks and brackets such as "{}", "()". Then, we apply Part of Speech (POS) tagging to the extracted sentence. POS tagging is the process of marking up a word in a text as corresponding to a particular part of speech, such as noun, verb, adjective. NLP tools usually agree on the POS tags of nouns, and we find that POS tagger in NLTK [10] is especially suitable for our task. In NLTK, the noun is annotated by different POS tags [1] including NN (Noun, singular or mass), NNS (Noun, plural), NNP (Proper noun, singular), NNPS (Proper noun, plural). Fig. 4 shows the results for the tag definition sentence of *Matplotlib*. Based on the POS tagging results, we extract the first noun (*library* in this example) after the be verb (*is* in this example) as the category of the tag. That is, the category of *Matplotlib* is *library*. Note that if the noun is some specific words such as *system*, *development*, we will further check its neighborhood words to see if it is *operating system* or *independent development environment*.

With this method, we obtain 318 categories for the 23,658 tags (about 67% of all the tags that have TagWiki). We manually normalize these 318 categories labels, such as merging *app* and *applications* as *application*, *libraries* and *lib* as *library*, and normalizing uppercase and lowercase (e.g., *API* and *api*). As a result, we obtain 167 categories. Furthermore, we manually categorize these 167 categories into five general categories: programming language, platform, library, API, and concept/standard [48]. This is because the meaning of the fine-grained categories is often overlapping, and there is no consistent rule for the usage of these terms in the TagWiki. This generalization step is necessary, especially for the library tags that broadly refer to the tags whose fine-grained categories can be library, framework, api, toolkit, wrapper, and so on. For example, in Stack Overflow's TagWiki, *junit* is defined as a framework, *google-visualization* is defined as an API, and *wxpython* is defined as a wrapper. All these tags are referred to as library tags in our approach.

Although the above method obtains the tag category for the majority of the tags, the first sentence of the TagWiki of some tags is not formatted in the standard "tag be noun phrase" form. For example, the first sentence of the TagWiki of the tag *itext* is "Library to create and manipulate PDF documents in Java", or for *markermanager*, the tag definition sentence is "A Google Maps tool", or for *ghc-pkg*, the tag definition sentence is "The command ghc-pkg can be used to handle GHC packages". As there is no *be* verb in this sentence, the above NLP method cannot return a noun phrase as the tag category. According to our observation, for most of such cases, the category of the tag is still present in the sentence, but often in many different ways. It is very likely that the category word appears as the first noun phrase that match the existing category words in the definition sentence. Therefore, we use a dictionary look-up method to determine the category of such tags. Specially, we use the 167 categories obtained using the above NLP method as a dictionary to recognize the category of the tags that have not been categorized using the NLP method. Given an uncategorized tag, we scan the first sentence of the tag's TagWiki from the beginning, and search for the first match of a category label in the sentence. If a match is found, the tag is categorized as the matched category. For example, the tag *itext* is categorized as *library* using this dictionary look-up method. Using the dictionary look-up method, we obtain the category for 9,648 more tags.

Note that we cannot categorize some (less than 15%) of the tags using the above NLP method and the dictionary look-up method. This is because these tags do not have a clear tag definition sentence, for example, the TagWiki of the tag *richtextbox* states that "The RichTextBox control enables you to display or edit RTF content". This sentence is not a clear definition of what *richtextbox* is. Or no category match can be found in the tag definition sentence of some tags. For example, the TagWiki of the tag *carousel* states that "A rotating display of content that can house a variety of content". Unfortunately, we do not have the category "display" in the 167 categories we collect using the NLP method. When building comparable-technologies knowledge base, we exclude these uncategorized tags as potential candidates.

Table 1: Examples of filtering results by categorical knowledge (in red)

Source	Top-5 recommendations from word embedding
nltk	~~nlp~~, opennlp, gate, ~~language-model~~, stanford-nlp
tcp	tcp-ip, ~~network-programming~~, udp, ~~packets~~, ~~tcpserver~~
vim	sublimetext, ~~vim-plugin~~, emacs, nano, gedit
swift	objective-c, ~~cocoa-touch~~, ~~storyboard~~, ~~launch-screen~~
bubble-sort	insertion-sort, selection-sort, mergesort, timsort, heapsort

2.3 Building Similar-technology Knowledge Base

Given a technology tag t_1 with its vector $vec(t_1)$, we first find most similar library t_2 whose vector $vec(t_2)$ is most closed to it, i.e.,

$$\underset{t_2 \in T}{\text{argmax}} \cos(vec(t_1), vec(t_2)) \quad (3)$$

where T is the set of technology tags excluding t_1, and $cos(u, v)$ is the cosine similarity of the two vectors.

Note that tags whose tag embedding is similar to the vector $vec(t_1)$ may not always be in the same category. For example, tag embeddings of the tags *nlp*, *language-model* are similar to the vector $vec(nltk)$. These tags are relevant to the *nltk* library as they refer to some NLP concepts and tasks, but they are not comparable libraries to the *nltk*. In our approach, we rely on the category of tags (i.e., categorical knowledge) to return only tags within the same category as candidates. Some examples can be seen in Table 1.

In practice, there could be several comparable technologies t_2 to the technology t_1. Thus, we select tags t_2 with the cosine similarity in Eq. 3 above a threshold *Thresh*. Take the library *nltk* (a NLP library in python) as an example. We will preserve several candidates which are libraries such as *textblob*, *stanford-nlp*.

3 MINING COMPARATIVE OPINIONS

For each pair of comparable technologies in the knowledge base, we analyze the Q&A discussions in Stack Overflow to extract plausible comparative sentences by which Stack Overflow users express their opinions on the comparable technologies. We may obtain many comparative sentences for each pair of comparable technologies. Displaying all these sentences as a whole may make it difficult for developers to read and digest the comparison information. Therefore, we measure the similarity among the comparative sentences, and then cluster them into several groups, each of which may identify a prominent aspect of technology comparison that users are concerned with.

3.1 Extracting Comparative Sentences

There are three steps to extract comparative sentences of the two technologies. We first carry out some preprocessing of the Stack Overflow post content. Then we locate the sentences that contain the name of the two technologies, and further select the comparative sentences that satisfy a set of comparative sentence patterns.

3.1.1 Preprocessing. To extract trustworthy opinions about the comparison of technologies, we consider only answer posts with positive score points. Then we split the textual content of such answer posts into individual sentences by punctuations like ".", "!", "?". We remove all sentences ended with question mark, as we want

Table 2: The 6 comparative sentence patterns

No.	Pattern	Sequence example	Original sentence
1	*TECH * VBZ * JJR*	innodb has 30 higher	InnoDB has 30% higher performance than MyISAM on average.
2	*TECH * VBZ * RBR*	postgresql is a more	Postgresql is a more correct database implementation while mysql is less compliant.
3	*JJR * CIN * TECH*	faster than coalesce	Isnull is faster than coalesce.
4	*RBR JJ * CIN TECH*	more powerful than velocity	Freemarker is more powerful than velocity.
5	*CV * CIN TECH*	prefer ant over maven	I prefer ant over maven personally.
6	*CV VBG TECH*	recommend using html5lib	I strongly recommend using html5lib instead of beautifulsoup.

Table 3: Examples of alias

Tech term	Synonyms	Abbreviation
visual studio	visualstudio, visual studios, visual-studio	msvs
beautifulsoup	beautiful soup	bs4
objective-c	objectivec, objective c	objc, obj-c
depth-first search	deep first search, depth first search, depth-first-search	dfs
postgresql	postgre sql, posgresq, postgesql	pgsql

to extract facts instead of doubts. We lowercase all sentences to make the sentence tokens consistent with the technology names because all tags are in lowercase.

3.1.2 Locating Candidate Sentences. To locate sentences mentioning a pair of comparable technologies, using only the tag names is not enough. As posts in Stack Overflow are informal discussions about programming-related issues, users often use alias to refer to the same technology [16]. Aliases of technologies can be abbreviations, synonyms and some frequent misspellings. For example, "javascript" are often written in many forms such as "js" (abbreviation), "java-script" (synonym), "javascrip" (misspelling) in the discussions.

The presence of such aliases will lead to significant missing of comparative sentences, if we match technology mentions in a sentence with only tag names. Chen et al.'s work [17] builds a large thesaurus of morphological forms of software-specific terms, including abbreviations, synonyms and misspellings. Table 3 shows some examples of technologies aliases in this thesaurus. Based on this thesaurus, we find 7310 different alias for 3731 software technologies. These aliases help to locate more candidate comparative sentences that mention certain technologies.

3.1.3 Selecting Comparative Sentences. To identify comparative sentences from candidate sentences, we develop a set of comparative sentence patterns. Each comparative sentence pattern is a sequence of POS tags. For example, the sequence of POS tags "*RBR JJ IN*" is a pattern that consists of a comparative adverb (*RBR*), an adjective (*JJ*) and subsequently a preposition (*IN*), such as "more efficient than", "less friendly than", etc. We extend the list of common POS tags to enhance the identification of comparative sentences. More specifically, we create three comparative POS tags: *CV* (comparative verbs, e.g. prefer, compare, beat), *CIN* (comparative prepositions, e.g. than, over), and *TECH* (technology reference, including the name and aliases of a technology, e.g. python, eclipse).

Based on data observations of comparative sentences, we summarise six comparative patterns. Table 2 shows these patterns and the corresponding examples of comparative sentences. To make the patterns more flexible, we use a wildcard character to represent a list of arbitrary words to match the pattern. For each sentence mentioning the two comparable technologies, we obtain its POS tags and check if it matches any one of six patterns. If so, the sentence will be selected as a comparative sentence.

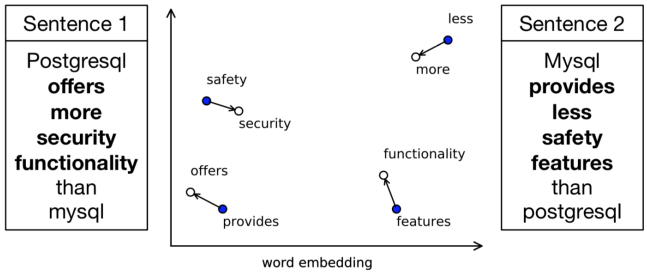


Figure 5: An illustration of measuring similarity of two comparative sentences

3.2 Measure Sentence Similarity

To measure the similarity of two comparative sentences, we adopt the Word Mover's Distance [28] which is especially useful for short-text comparison. Given two sentences S_1 and S_2, we take one word i from S_1 and one word j from S_2. Let their word vectors be v_i and v_j. The distance between the word i and the word j is the Euclidean distance between their vectors, $c(i, j) = ||v_i - v_j||_2$. To avoid confusion between word and sentence distance, we will refer to $c(i, j)$ as the cost associated with "traveling" from one word to another. One word i in S_1 may move to several different words in the S_2, but its total weight is 1. So we use $T_{ij} \geq 0$ to denote how much of word i in S_1 travels to word j in S_2. It costs $\sum_j T_{ij}c(i, j)$ to move one word i entirely into S_2. We define the distance between the two sentences as the minimum (weighted) cumulative cost required to move all words from S_1 to S_2, i.e., $D(S_1, S_2) = \sum_{i,j} T_{ij}c(i, j)$. This problem is very similar to transportation problem i.e., how to spend less to transform all goods from source cities $A_1, A_2, \ldots$ to target cities $B_1, B_2, \ldots$. Getting such minimum cost actually is a well-studied optimization problem of earth mover distance [32, 38].

To use word mover's distance in our approach, we first train a word embedding model based on the post content of Stack Overflow so that we get a dense vector representation for each word in Stack Overflow. Word embedding has been shown to be able to capture rich semantic and syntactic information of words. Our approach does not consider word mover's distance for all words in a sentence. Instead, for each comparative sentence, we extract only keywords with POS tags that are most relevant to the comparison, including adjectives (JJ), comparative adjectives (JJR) and nouns (NN, NNS, NNP and NNPS), not including the technologies under comparison. Then, we compute the minimal word movers' distance between the keywords in one sentence and those in the other sentences. Base on the distance, we further compute the similarity score of the two sentences by

$$similarity\ score(S_1, S_2) = \frac{1}{1 + D(S_1, S_2)}$$

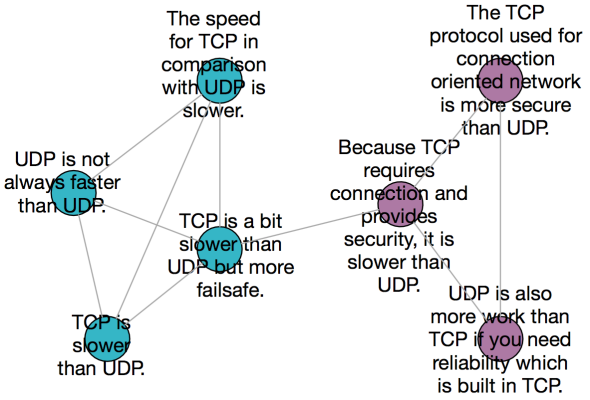


Figure 6: Communities in the graph of comparative sentences

The similarity score is in range $(0, 1)$, and the higher the score, the more similar the two sentences. If the similarity score between the two sentences is larger than the threshold, we regard them as similar. The threshold is 0.55 in this work, determined heuristically by a small-scale pilot study. We show some similar comparative sentences by word mover's distance in Table 4.

To help reader understand word movers' distance, we show an example in Figure 5 with two comparative sentences for comparing *postgresql* and *mysql*: "*Postgresql offers more security functionality than mysql*" and "'*Mysql provides less safety features than postgresql*'. The keywords in the two sentences that are most relevant to the comparison are highlighted in bold. We see that the minimum distance between the two sentences is mainly the accumulation of word distance between pairs of similar words *(offers, provides)*, *(more, less)*, *(security, safety)*, and *(functionality, features)*. As the distance between the two sentences is small, the similarity score is high even though the two sentences use rather different words and express the comparison in reverse directions.

3.3 Clustering Representative Comparison Aspects

For each pair of comparable technologies, we collect a set of comparative sentences about their comparison in Section 3.1. Within these comparative sentences, we find pairs of similar sentences in Section 3.2. We take each comparative sentence as one node in the graph. If the two sentences are determined as similar, we add an edge between them in the graph. In this way, we obtain a graph of comparative sentences for a given pair of comparative technologies.

Although some comparative sentences are very different in words or comparison directions (examples shown in Fig. 5 and Table 4), they may still share the same comparison opinions. In graph theory, a set of highly correlated nodes is referred to as a community (cluster) in the network. Based on the sentence similarity, we cluster similar opinions by applying the community detection algorithm to the graph of comparative sentences. In this work, we use the Girvan-Newman algorithm [21] which is a hierarchical community detection method. It uses an iterative modularity maximization method to partition the network into a finite number of disjoint clusters that will be considered as communities. Each node must be assigned to exactly one community. Fig. 6 shows the graph of comparative sentences for the comparison of *TCP* and *UDP* (two network protocols), in which each node is a comparative sentence, and the detected communities are visualized in the same color.

As seen in Fig. 6, each community may represent a prominent comparison aspect of the two comparable technologies. But some communities may contain too many comparative sentences to understand easily. Therefore, we use TF-IDF (Term Frequency Inverse Document Frequency) to extract keywords from comparative sentence in one community to represent the comparison aspect of this community. TF-IDF is a statistical measure to evaluate the importance of a word to a document in a collection. It consists of two parts: term frequency (TF, the number occurrences of a term in a document) and inverse document frequency (IDF, the logarithm of the total number of documents in the collection divided by the number of documents in the collection that contain the specific term). For each community, we remove stop words in the sentences, and regard each community as a document. We take the top-3 words with largest TF-IDF scores as the representative aspect for the community. Table 5 shows the comparison aspects of four communities for comparing *postgresql* with *mysql*. The representative keywords directly show that the comparison between *postgresql* with *mysql* mainly focuses on four aspects: speed, security, popularity, and usability.

4 IMPLEMENTATION

4.1 Dataset

We take the latest Stack Overflow data dump (released on 13 March 2018) as the data source. It contains 14,995,834 questions, 23,399,083 answers, 50,812 unique tags. With the approach in Section 2, we collect in total 14,876 pairs of comparable technologies. Among these technologies, we extract 14,552 comparative sentences for 2,074 pairs of comparable technologies. We use these technologies and comparative sentences to build a knowledge base for technology comparison.

4.2 Tool Support

Apart from our abstract approach, we also implement a practical tool[2] for developers. With the knowledge base of comparable technologies and their comparative sentences mined from Stack Overflow, our site can return an informative and aggregated view of comparative sentences in different comparison aspects for comparable technology queries. In addition, the tool provides the link of each comparative sentence to its corresponding Stack Overflow post so that users can easily find more detailed content.

5 EXPERIMENT

In this section, we evaluate each step of our approach. As there is no ground truth for technology comparison, we have to manually check the results of each step or build the ground truth. And as it is clear to judge whether a tag is of a certain category from its tag description, whether two technologies are comparable, and whether a sentence is a comparative sentence, we recruit two Master students

[2] https://difftech.herokuapp.com/

Table 4: Examples of similar comparative sentences by Word Mover's Distance

Comparable technology pair	Comparative sentences
vmware & virtualbox	Virtualbox is slower than vmware. In my experience I've found that vmware seems to be faster than virtualbox.
strncpy & strcpy	In general strncpy is a safer alternative to strcpy. So that the strncpy is more secure than strcpy.
google-chrome & safari	Safari still uses the older Webkit while Chrome uses a more current one. Google Chrome also uses an earlier version of Webkit than Safari.
quicksort & mergesort	Mergesort would use more space than quicksort. Quicksort is done in place and doesn't require allocating memory, unlike mergesort.
nginx & apache	Serving static files with nginx is much more efficient than with apache. There seems to be a consensus that nginx serves static content faster than apache.

Table 5: The representative keywords for clusters of *postgresql* and *mysql*.

Representative keywords	Comparative sentences
speed, slower, faster	In most regards, postgresql is slower than mysql especially when it comes to fine tuning in the end. I did a simple performance test and I noticed postgresql is slower than mysql. According to my own experience, postgresql run much faster than mysql. Postgresql seem to better than mysql in terms of speed.
security, safety, functionality	Traditionally postgresql has fewer security issues than mysql. Postgresql offers more security functionality than mysql. Mysql provides less safety features than postgresql.
popular	While postgresql is less popular than mysql, most of the serious web hosting supports it. Though mysql is more popular than postgresql but instagram is using postgresql maybe due to these reasons. It's a shame postgresql isn't more popular than mysql, since it supports exactly this feature out-of-the-box.
easier, simplicity	Mysql is more widely supported and a little easier to use than postgresql. Postgresql specifically has gotten easier to manage while mysql has lost some of the simplicity. However, people often argue that postgresql is easier to use than mysql.

to manually check the results of these three steps. Only results that they both agree will be regarded as ground truth for computing relevant accuracy metrics, and those results without consensus will be given to the third judge who is a PhD student with more experience. All three students are majoring in computer science and computer engineering in our school, and they have diverse research and engineering background with different software tools and programming languages in their work. In addition, we release all experiment data and results in our website[3].

5.1 Accuracy of Extracting Comparable Technologies

This section reports our evaluation of the accuracy of tag category identification, the important of tag category for filtering out irrelevant technologies, and the impact of word embedding models and hyperparameters.

5.1.1 The Accuracy of Tag CCategory. From 33,306 tags with tag category extracted by our method, we randomly sample 1000 tags whose categories are determined using the NLP method, and the other 1000 tags whose categories are determined by the dictionary look-up method (see Section 2.2). Among the 1000 sampled tag categories by the NLP method, categories of 838 (83.8%) tags are correctly extracted by the proposed method. For the 1000 sampled tags by the dictionary look-up method, categories of 788 (78.8%) tags are correct.

According to our observation, two reasons lead to the erroneous tag categories. First, some tag definition sentences are complex which can lead to erroneous POS tagging results. For example, the tagWiki of the tag *rpy2* states that "RPy is a very simple, yet robust, Python interface to the R Programming Language". The default POS tagging recognizes *simple* as the noun which is then regarded as the category by our method. Second, the dictionary look-up method sometimes makes mistakes, as the matched category may not be the real category. For example, the TagWiki of the tag *honeypot* states "A trap set to detect or deflect attempts to hack a site or system". Our approach matches the *system* as the category of the *honeypot*.

5.1.2 The Importance of Tag Category. To check the importance of tag category for the accurate comparable technology extraction, we set up two methods, i.e., one is word embedding and tag category filtering, and the other is only with word embedding. The word embedding model in two methods are both skip-gram model with the word embedding dimension as 800. We randomly sample 150 technologies pairs extracted from each method, and manually check the if the extracted technology pair is comparable or not. It shows that the performance of model with tag category (90.7%) is much better than that without the tag category filtering (29.3%).

5.1.3 The impact of parameters of word embedding. There are two important parameters for the word embedding model, and we test

[3] https://sites.google.com/view/difftech/

Table 6: The accuracy of comparative sentences extraction

No.	Pattern	#right	#wrong	Accuracy
1	*TECH * VBZ * JJR*	44	6	88%
2	*TECH * VBZ * RBR*	45	5	90%
3	*JJR * CIN * TECH*	43	7	86%
4	*RBR JJ * CIN TECH*	47	3	94%
5	*CV * CIN TECH*	37	13	74%
6	*CV VBG TECH*	35	15	70%
Total		251	49	83.7%

its impact on the the performance of our method. First, we compare the performance of CBOW and Skip-gram mentioned in Section 2.1 by randomly sampling 150 technology pairs extracted by each method under the same parameter setting (the word embedding dimension is 400). The results show that Skip-gram model (90.7%) outperforms the CBOW model (88.7%), but the difference is marginal.

Second, we randomly sample 150 technologies pairs by the skip-gram model with different word embedding dimensions, and manually check the accuracy. From the dimension 200 to 1000 with the step as 200, the accuracy is 70.7%, 72.7%, 81.3%, 90.7%, 87.3%. We can see that the model with the word embedding dimension as 800 achieves the best performance. Finally, we take the Skip-gram model with 800 word-embedding dimension as the word embedding model to obtain the comparable technologies in this work.

5.2 Accuracy and coverage of comparative sentences

We evaluate the accuracy and coverage of our approach in finding comparative sentences from the corpus. We first randomly sample 300 sentences (50 sentences for each comparative sentence pattern in Table 2) which are extracted by our model. We manually check the accuracy of the sampled sentences and Table 6 shows the results. The overall accuracy of comparative sentence extraction is 83.7%, and our approach is especially accurate for the first 4 patterns. The last two patterns do not get good performance due to the relatively loose conditions.

We further check the wrong extraction of comparative sentences and find that most errors are caused by wrong comparable technologies extracted in Section 2. For example, *implode* and *explode* are not comparable technologies, but they are mentioned in sentence "I'm not sure why you'd serialize it in php either because *implode* and *explode* would be more appropriate". In addition, although some sentences do not contain the question mark, they are actually interrogative sentence such as "*I also wonder if postgresql will be a win over mysql*".

5.3 Accuracy of clustering comparative sentences

We evaluate the performance of our opinion clustering method by comparing it with the baseline methods.

5.3.1 Baseline. We set up two baselines to compare with our comparative sentence clustering method. The first baseline is the traditional TF-IDF [40] with K-means [23], and the second baseline is based on the document-to-vector deep learning model (i.e., Doc2vec [29]) with K-means. Both methods first convert the comparative sentences for a pair of comparable technologies into vectors by TF-IDF and Doc2vec. Then for both methods, we carry out K-means algorithms to cluster the sentence vectors into N clusters. To make the baseline as competitive as possible, we set N at the cluster number of the ground truth. In contrast, our method specifies its cluster number by community detection which may differ from the cluster number of the ground truth.

Table 7: Ground Truth for evaluating clustering results

No.	Technology pair	#comparative sentences	#clusters
1	*compiled & interpreted language*	27	5
2	*sortedlist & sorteddictionary*	11	4
3	*ant & maven*	47	7
4	*pypy & cpython*	51	3
5	*google-chrome & safari*	35	6
6	*quicksort & mergesort*	54	4
7	*lxml & beautifulsoup*	32	4
8	*awt & swing*	30	3
9	*jackson & gson*	31	3
10	*swift & objective-c*	72	10
11	*jruby & mri*	19	3
12	*memmove & memcpy*	21	3

5.3.2 Ground Truth. As there is no ground truth for clustering comparative sentences, we ask two Master students mentioned before to manually build a small-scale ground truth. We randomly sample 15 pairs of comparable technologies with different number of comparative sentences. For each technology pair, the two students read each comparative sentence and each of them will individually create several clusters for these comparative sentences. Note some comparative sentences are unique without any similar comparative sentence, and we put all those sentences into one cluster. Then they will discuss with the Ph.D student about the clustering results, and change the clusters accordingly. Finally, they reach an agreement for 12 pairs of comparable technologies. We take these 12 pairs as the ground truth whose details can be seen in Table 7.

5.3.3 Evaluation Metrics. Given the ground truth clusters, many metrics have been proposed to evaluate the clustering performance in the literature. In this work, we take the Adjusted Rand Index (ARI) [25], Normalized Mutual Information(NMI) [47], homogeneity, completeness, V-measure [39], and Fowlkes-Mallows Index (FMI) [20]. For all six metrics, higher value represents better clustering performance. For each pair of comparable technologies, we take all comparative sentences as a fixed list, and G as a ground truth cluster assignment and C as the algorithm clustering assignment.

Adjusted Rand Index (ARI) measures the similarity between two partitions in a statistical way. It first calculates the raw Rand Index (RI) by $RI = \frac{a+b}{C_2^N}$ where a is the number of pairs of elements that are in the same cluster in G and also in the same cluster in C, and b is the number of pairs of elements that are in different clusters in G and also in different clusters in C. C_2^N is the total number of possible pairs in the dataset (without ordering) where N is the number of comparative sentences. To guarantee that random label assignments will get a value close to zero, ARI is defined as

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

where $E[RI]$ is the expected value of RI.

Normalized Mutual Information (NMI) measures the mutual information between the ground truth labels G and the algorithm clustering labels C, followed by a normalization operation:

$$NMI(G,C) = \frac{MI(G,C)}{\sqrt{H(G)H(C)}}$$

where $H(G)$ is the entropy of set G i.e., $H(G) = -\sum_{i=1}^{|G|} P(i)\log(P(i))$ and $P(i) = \frac{G_i}{N}$ is the probability than an objet picked at random falls into class G_i. The $MI(G,C)$ is the mutual information between G and C where $MI(G,C) = \sum_{i=1}^{|G|}\sum_{j=1}^{|C|} P(i,j)\log(\frac{P(i,j)}{P(i)P(j)})$

Homogeneity (HOM) is the proportion of clusters containing only members of a single class by

$$h = 1 - \frac{H(G|C)}{H(G)}$$

Completeness (COM) is the proportion of all members of a given class are assigned to the same cluster by

$$c = 1 - \frac{H(C|G)}{H(C)}$$

where $H(G|C)$ is the conditional entropy of the ground-truth classes given the algorithm clustering assignments.

V-measure (V-M) is the harmonic mean of homogeneity and completeness

$$v = 2 \times \frac{h \times c}{h + c}$$

Fowlkes-Mallows Index (FMI) is defined as the geometric mean of the pairwise precision and recall:

$$FMI = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}}$$

where TP is the number of True Positive (i.e. the number of pairs of comparative sentences that belong to the same clusters in both the ground truth and the algorithm prediction), FP is the number of False Positive (i.e. the number of pairs of comparative sentences that belong to the same clusters in the ground-truth labels but not in the algorithm prediction) and FN is the number of False Negative (i.e the number of pairs of comparative sentences that belongs in the same clusters in the algorithm prediction but not in the ground truth labels).

5.3.4 Overall Performance. Table 8 shows the evaluation results. TF-IDF with K-means has similar performance as the Doc2vec with K-means, but our model significantly outperforms both models in all six metrics.

According to our inspection of detailed results, we find two reasons why our model outperforms two baselines. First, our model can capture the semantic meaning of comparative sentences. TF-IDF can only find similar sentences using the same words but count similar words like "secure" and "safe" as unrelated. While the sentence vector from Doc2vec is easily influenced by the noise as it takes all words in the sentence into consideration. Second, constructing the similar sentences as a graph in our model explicitly encodes the sentence relationships. The community detection based on the graph can then easily put similar sentences into clusters. In contrast, for the two baselines, the error brought from the TF-IDF and Doc2vec is accumulated and amplified to K-means in the clustering phase.

Table 8: Clustering performance

Method	ARI	NMI	HOM	COM	V-M	FMI
TF-IDF+Kmeans	0.12	0.28	0.29	0.27	0.28	0.41
Doc2vec+Kmeans	-0.01	0.11	0.10	0.14	0.11	0.43
Our model	0.66	0.73	0.75	0.72	0.73	0.79

6 USEFULNESS EVALUATION

Experiments in Section 5 have shown the accuracy of our approach. In this section, we further demonstrate the usefulness of our approach. According to our observation of Stack Overflow, there are some questions discussing comparable technologies such as "*What is the difference between Swing and AWT*". We demonstrate the usefulness of the technology-comparison knowledge our approach distills from Stack Overflow discussions by checking how well the distilled knowledge by our approach can answer those questions.

6.1 Evaluation Procedures

We use the name of comparable technologies with several keywords such as *compare*, *vs*, *difference* to search questions in Stack Overflow. We then manually check which of them are truly about comparable technology comparison, and randomly sample five questions that discuss comparable technologies in different categories and have at least five answers. The testing dataset can be seen in Table 9.

We then ask the two Master students to read each sentence in all answers and cluster all sentences into several clusters which represent developers' opinions in different aspects. To make the data as valid as possible, they still first carry out the clustering individually and then reach an agreement after discussions. For each comparative opinion in the answer, we manually check if that opinion also appears in the knowledge base of comparative sentences extracted by our method. To make this study fair, our method does not extract comparative sentences from answers of questions used in this experiment.

6.2 Results

Table 10 shows the evaluation results. We can see that most comparison (72%) aspects can be covered by our knowledge base. For two questions (#5970383 and #46585), the technology-comparison knowledge distilled by our method can cover all of comparison aspects in the original answers such as speed, reliability, data size for comparing *post* and *get*. While for the other three questions, our model can still cover more than half of the comparison aspects.

We miss some comparison aspects for the other three questions, such as "*One psychological reason that has not been given is simply that Quicksort is more cleverly named, i.e. good marketing.*", "*The VMWare Workstation client provides a nicer end-user experience (subjective, I know...)*" and "*Another statement which I saw is that swing is MVC based and awt is not.*". Such opinions are either too subjective or too detailed which rarely appear again in other Stack Overflow discussions, leading to not having them in our knowledge base.

Apart from comparison aspects appeared in the original answers, our tool can provide some unique opinions from other aspects, such

Table 9: Comparative questions

Question ID	Question title	Tech pair	Tech category	#answers
70402	Why is quicksort better than mergesort?	*quicksort & mergesort*	Algorithm	29
5970383	Difference between TCP and UDP	*tcp & udp*	Protocol	9
630179	Benchmark: VMware vs Virtualbox	*vmware & virtualbox*	IDE	13
408820	What is the difference between Swing and AWT?	*swing & awt*	Library	8
46585	When do you use POST and when do you use GET?	*post & get*	Method	28

Table 10: Distilled knowledge by our approach versus original answers

Question ID	#Aspects	#Covered	#Unique in our model
70402	6	4 (66.67%)	2
5970383	3	3 (100%)	5
630179	7	4 (57.1%)	1
408820	5	3 (60%)	4
46585	4	4 (100%)	2
Total	25	18 (72%)	14

as "*In my experience, udp based code is generally less complex than tcp based code*" for comparing *tcp* and *udp*, "*however I found that vmware is much more stable in full screen resolution to handle the iphone connection via usb*" for comparing *vmware* and *virtualbox*, and "*GET would obviously allow for a user to change the value a lot easier than POST*" for comparing *post* and *get*. As seen in Table 10, our model can provide more than one unique comparative aspects which are not in the existing answers for each technology pair. Therefore, our knowledge base can be a good complement to these existing technology-comparison questions with answers. Furthermore, our knowledge base contains the comparison knowledge of 2074 pairs of comparable technologies, many of which have not been explicitly asked and discussed in Stack Overflow, such as *swift* and *objective-c*, *nginx* and *apache*.

7 RELATED WORKS

Finding similar software artefacts can help developers migrate from one tool to the other which is more suitable to their requirement. But it is a challenging task to identify similar software artefacts from the existing large pool of candidates. Therefore, much research effort has been put into this domain. Different methods has been adopted to mine similar artefacts ranging from high-level software [33, 43], mobile applications [19, 31], github projects[49] to low-level third-party libraries [11, 13, 42], APIs [22, 37], code snippets [41], or Q&A questions [18]. Compared with these research studies, the mined software technologies in this work has much broader scope including not only software-specific artefacts, but also general software concepts (e.g., algorithm, protocol), tools (e.g., IDE).

Given a list of similar technologies, developers may further compare and contrast them for the final selection. Some researcher investigate such comparison, the comparison is highly domain-specific such as software for traffic simulation [26], regression models [24], x86 virtualization [7], etc. Michail and Notkin [34] assess different third-party libraries by matching similar components (such as classes and functions) across similar libraries. But it can only work for library comparison without the possibility to be extended to other higher/lower-level technologies in Software Engineering. Instead, we find developers's preference of certain software technologies highly depends on other developers' usage experience and report of similar technology comparisons. Therefore, Uddin and Khomh [45, 46] extract API opinion sentences in different aspects to show developers' sentiment to that API. Li et al. [30] adopt NLP methods to distill comparative user review about similar mobile Apps. Different from their works, we first explicitly extract a large pool of comparable technologies. In addition, apart from extracting comparative sentences, we further organize them into different clusters and represent each cluster with some keywords to help developers understand comparative opinions more easily.

Finally, it is worth mentioning some related practical projects. SimilarWeb [6] is a website that provides both users engagement statistics and similar competitors for websites and mobile applications. AlternativeTo [4] is a social software recommendation website in which users can find alternatives to a given software based on user recommendations. SimilarTech [5] is a site to recommend analogical third-party libraries across different programming languages. These websites can help users find similar or alternative websites or software applications without detailed comparison.

8 CONCLUSION AND FUTURE WORK

In this paper, we present an automatic approach to distill and aggregate comparative opinions of comparable technologies from Q&A websites. We first obtain a large pool of comparable technologies by incorporating categorical knowledge into word embedding of tags in Stack Overflow, and then locate comparative sentences about these technologies by POS-tag based pattern matching, and finally organize comparative sentences into clusters for easier understanding. The evaluation shows that our system covers a large set of comparable technologies and their corresponding comparative sentences with high accuracy. We also demonstrate the potential of our system to answer questions about comparing comparable technologies, because the technology comparison knowledge mined using our system largely overlap with the original answers in Stack Overflow.

Apart from comparative sentences explicitly mentioning both comparable technologies, some comparative opinions may hide deeper. For example, one developer expresses his opinions about one technology in one paragraph while discussing the other technology in the next paragraph. Therefore, we will improve our system to distill technology comparison knowledge from the current sentence level to post level. In addition, we also plan to summarize higher-level opinions or preferences from separated individual comparative sentences for easier understanding.

REFERENCES

[1] 2003. Alphabetical list of part-of-speech tags used in the Penn Treebank Project. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html. Accessed: 2018-02-02.

[2] 2017. Millions of Queries per Second: PostgreSQL and MySQLâĂŹs Peaceful Battle at TodayâĂŹs Demanding Workloads. https://goo.gl/RXVjkB/. Accessed: 2018-04-05.

[3] 2017. MySQL vs Postgres. https://www.upguard.com/articles/postgres-vs-mysql/. Accessed: 2018-04-05.

[4] 2018. AlternativeTo - Crowdsourced software recommendations. https://alternativeto.net/. Accessed: 2018-04-05.

[5] 2018. SimilarTech: Find alternative libraries across languages. https://graphofknowledge.appspot.com/similartech/. Accessed: 2018-04-05.

[6] 2018. SimilarWeb. https://www.similarweb.com/. Accessed: 2018-04-05.

[7] Keith Adams and Ole Agesen. 2006. A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Computer Architecture News* 34, 5 (2006), 2–13.

[8] Lingfeng Bao, Jing Li, Zhenchang Xing, Xinyu Wang, Xin Xia, and Bo Zhou. 2017. Extracting and analyzing time-series HCI data from screen-captured task videos. *Empirical Software Engineering* 22, 1 (2017), 134–174.

[9] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[10] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.

[11] Chunyang Chen, Sa Gao, and Zhenchang Xing. 2016. Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, Vol. 1. IEEE, 338–348.

[12] Chunyang Chen and Zhenchang Xing. 2016. Mining technology landscape from stack overflow. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 14.

[13] Chunyang Chen and Zhenchang Xing. 2016. Similartech: automatically recommend analogical libraries across different programming languages. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 834–839.

[14] Chunyang Chen and Zhenchang Xing. 2016. Towards correlating search on google and asking on stack overflow. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, Vol. 1. IEEE, 83–92.

[15] Chunyang Chen, Zhenchang Xing, and Lei Han. 2016. Techland: Assisting technology landscape inquiries with insights from stack overflow. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 356–366.

[16] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2017. By the Community & For the Community: A Deep Learning Approach to Assist Collaborative Editing in Q&A Sites. *Proceedings of the ACM on Human-Computer Interaction* 1, 32 (2017), 1–32.

[17] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 450–461.

[18] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 744–755.

[19] Ning Chen, Steven CH Hoi, Shaohua Li, and Xiaokui Xiao. 2015. SimApp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. ACM, 305–314.

[20] Edward B Fowlkes and Colin L Mallows. 1983. A method for comparing two hierarchical clusterings. *Journal of the American statistical association* 78, 383 (1983), 553–569.

[21] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[22] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2017. DeepAM: Migrate APIs with multi-modal sequence to sequence learning. *arXiv preprint arXiv:1704.07734* (2017).

[23] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.

[24] Nicholas J Horton and Stuart R Lipsitz. 2001. Multiple imputation in practice: comparison of software packages for regression models with missing variables. *The American Statistician* 55, 3 (2001), 244–254.

[25] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[26] Steven L Jones, Andrew J Sullivan, Naveen Cheekoti, Michael D Anderson, and D Malave. 2004. Traffic simulation software comparison study. *UTCA report* 2217 (2004).

[27] JunâĂŹichi Kazama and Kentaro Torisawa. 2007. Exploiting Wikipedia as external knowledge for named entity recognition. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 698–707.

[28] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International Conference on Machine Learning*. 957–966.

[29] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[30] Yuanchun Li, Baoxiong Jia, Yao Guo, and Xiangqun Chen. 2017. Mining User Reviews for Mobile App Comparisons. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 75.

[31] Mario Linares-Vásquez, Andrew Holtzhauer, and Denys Poshyvanyk. 2016. On automatically detecting similar android apps. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 1–10.

[32] Haibin Ling and Kazunori Okada. 2007. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE transactions on pattern analysis and machine intelligence* 29, 5 (2007), 840–853.

[33] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. 2012. Detecting similar software applications. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 364–374.

[34] Amir Michail and David Notkin. 1999. Assessing software libraries by browsing similar classes, functions and relationships. In *Proceedings of the 21st international conference on Software engineering*. ACM, 463–472.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[36] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[37] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N Nguyen. 2017. Exploring API embedding for API usages and applications. In *Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on*. IEEE, 438–449.

[38] Ofir Pele and Michael Werman. 2009. Fast and robust earth mover's distances. In *Computer vision, 2009 IEEE 12th international conference on*. IEEE, 460–467.

[39] Andrew Rosenberg and Julia Hirschberg. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*.

[40] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[41] Fang-Hsiang Su, Jonathan Bell, Gail Kaiser, and Simha Sethumadhavan. 2016. Identifying functionally similar code in complex codebases. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 1–10.

[42] Cédric Teyton, Jean-Rémy Falleri, and Xavier Blanc. 2013. Automatic discovery of function mappings between similar libraries. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 192–201.

[43] Ferdian Thung, David Lo, and Lingxiao Jiang. 2012. Detecting similar applications with collaborative tagging. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 600–603.

[44] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How do programmers ask and answer questions on the web?: Nier track. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 804–807.

[45] Gias Uddin and Foutse Khomh. 2017. Automatic summarization of API reviews. In *Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on*. IEEE, 159–170.

[46] Gias Uddin and Foutse Khomh. 2017. Opiner: an opinion search and summarization engine for APIs. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 978–983.

[47] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research* 11, Oct (2010), 2837–2854.

[48] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-specific named entity recognition in software engineering social content. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, Vol. 1. IEEE, 90–101.

[49] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. 2017. Detecting similar repositories on GitHub. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 13–23.