DiffTech: Differencing Similar Technologies from Crowd-Scale Comparison Discussions

Han Wang, Chunyang Chen, Zhenchang Xing, and John Grundy

Abstract—Developers use different technologies for many software development tasks. However, when faced with several technologies with comparable functionalities, it is not easy to select the most appropriate one, as trial and error comparisons among such technologies are time-consuming. Instead, developers can resort to expert articles, read official documents or ask questions in Q&A sites. However, it still remains difficult to get a comprehensive comparison as online information is often fragmented or contradictory. To overcome these limitations, we propose the DIFFTECH system that exploits crowdsourced discussions from Stack Overflow, and assists technology comparison with an informative summary of different aspects. We first build a large database of comparable technologies in software engineering by mining tags in Stack Overflow. We 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 and aggregate the overall opinion towards the comparable technologies. Our evaluation demonstrates both the accuracy and usefulness of our model, and we have implemented our approach as a practical website for public use.

Index Terms—comparing and differencing similar technology, Stack Overflow, natural language processing, NLP

1 INTRODUCTION

A diverse set of technologies – algorithms, programming languages, platforms, libraries/frameworks, concepts for software engineering [1], [2] – is available for use by developers, and this continues to grow significantly. Adopting the most suitable technologies for a problem will significantly accelerate software development and enhance the software quality [3]. When developers are looking for the right technology choices 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 suitable choice for their work. However, even for experienced developers, it can be challenging to keep pace with the rapid evolution of different software technologies. Developers can try each of the candidates in their work to compare them. Such a trial-and-error based 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. Thus, developers often turn to two key information sources on the Web [4] to learn more about comparable technologies.

First, they read experts' articles about technology comparison, such as *"Intellij vs. Eclipse: Why IDEA is Better"*.

- *Han Wang, Chunyang Chen (corresponding author), and John Grundy are with Faculty of Information Technology, Monash University, Australia. E-mail: freddie.wanah@gmail.com, chunyang.chen@monash.edu, john.grundy@monash.edu.*
- *Zhenchang Xing is with College of Engineering & Computer Science, Australian National University, Australia. E-mail: zhenchang.xing@anu.edu.au*

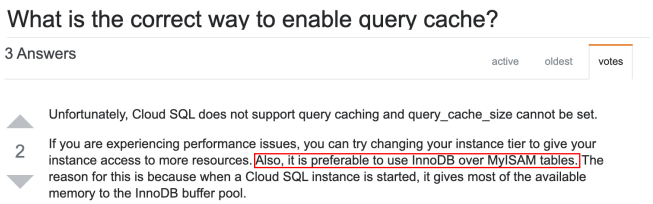


Fig. 1. A comparative sentence in a question (#30654296) that is not explicitly for technology comparison.

Second, developers 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 indexed by search engines, thus enabling developers to quickly find answers to their technology comparison inquiries.

However, there are two major limitations with using these 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 from different aspects. For example, to compare *mysql* and *postgresql*, one article [5] contrasts their speed, while another [6] compares their reliability. Only after reading both articles, developers may then have a more comprehensive overview of these two comparable technologies.
- *Diverse opinions:* Each expert article or community answer is based on the author's knowledge and experience. However, the knowledge and experience of developers vary greatly, and because they all work on vastly different projects, their perspective on a technology may also differ. For example, one may prefer *Eclipse* over *Intellij* because *Eclipse* fits his project setting better. But that setting may not be generalisable to other developers. At the same time, some developers may prefer *Intellij* over *Eclipse* for other reasons, including their own preferences and positive

or negative experiences. Such contradictory preferences among different opinions may confuse developers.

Our DIFFTECH system is motivated by the fact that a wide range of technologies has been discussed by millions of users in Stack Overflow [7], and users often express their preferences toward a technology and compare one technology with the others in these discussions. Apart from posts explicitly about the comparison of some technologies, many comparative sentences hide in posts are implicitly about technology comparisons. Fig.1 shows such an example: the answer "accidentally" compares *Innodb* and *Myisam*. The question "What is the correct way to enable query cache?" does not explicitly ask for this comparison. Inspired by such a phenomenon, we designed our DIFFTECH system to mine and aggregate such technology comparative sentences from Stack Overflow discussions.

As shown in Fig. 2, the input is a set of tags $\{T_1, T_2, ...T_n\}$ with their definitions in TagWiki, and posts data from Stack Overflow data dump. We consider Stack Overflow tags as a collection of technology terms and first find sets of comparable technology pairs $(T_1, T_2), (T_3, T_4), \ldots (T_i, T_j)...$ by analyzing tag embeddings and categories (1-3). DIFFTECH then mines comparative opinions from Q&A discussions by checking sentence patterns, clustering with word mover distance [8] and community detection [9]. Given a comparable technology pair (T_i, T_j), we mine comparable sentences $\{S_1, S_2, ...S_m\}$ that are related to (T_i, T_j) (4, 5). Based on extracted sentences, we further cluster them into a set of categories $\{C_1, C_2, \ldots C_p\}$ (6). Finally, we fine-tune the BERT [10] model to summarize overall sentiment of all comparative sentences for each comparable technology pairs (7). The outputs are comparable opinions of technologies. As a result, our approach takes a query Q, made up of a technology pair, $Q = \{T_1, T_2\}$, and maps Q onto a set of clustered comparable sentences, i.e., $f(Q) = \{C_1 : (S_1, S_2, ...), \ldots C_p : (..., S_i, S_j)\}$. There are summative opinions for each category C. We also implemented a practical website[1] for developers to compare similar technologies.

Our DIFFTECH is especially useful in some specific scenarios. First, when developers are trying to spend as little time as possible to figure out which technology is better. Since our DIFFTECH provides the overall preference from the crowd, developers can easily make the decision by reviewing our aggregated results. Second, when some novice developers or students are trying to learn the similarity and difference between similar technologies, our DIFFTECH is a good platform for their initial reference. But our tool may not be suitable for developers who are seeking for rigid comparison of similar technologies such as the performance of third-party libraries in their specific development context. Developers have to run formal performance experiments in their specific environment, though our DIFFTECH may provide them with initial inspiration.

As there is no ground truth for technology comparison, we manually validate the performance of each step of our approach. Our experimental results confirm the accuracy of comparable technology identification (90.7%), and distilling comparative sentences (88.8%) from Q&A discussions.

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

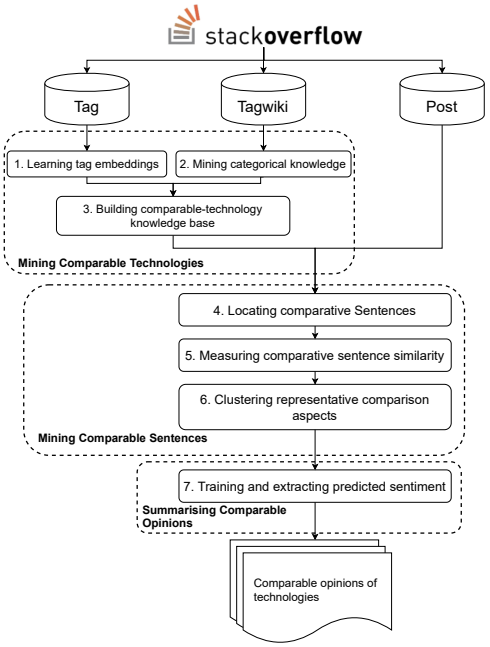


Fig. 2. Overview of our approach

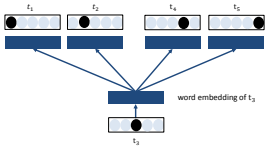
By manually building our ground truth, we show that our clustering method (word mover distance and community detection) for comparative sentences significantly outperforms other baselines. Regarding the accuracy of overall sentiment summarization, our model also provides 13.6% improvement over the baselines. We also demonstrate the generality of our approach by successfully extracting comparative opinions of comparable technologies in other domain-specific datasets.

This paper is an extension of our prior study [11], with the following additional contributions:

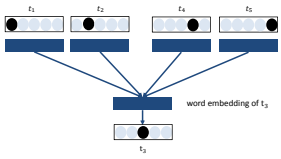
- We extend the single sentence patterns to take context and coreference into consideration to discover more comparative sentences about similar technologies. We also add the contextual sentences patterns to extract more sentences.
- To give developers a clearer overview, we develop a classifier for identifying the sentiment towards each technology, and aggregate crowd opinions in total.
- We have extended our previous experiments by including new data and new baselines, and carried out more detailed analysis of experimental results.
- We demonstrated the generality of our method by applying it to other domain-specific datasets, and demonstrated the usefulness of DiffTech via a user study.
- Based on the extracted comparative opinions, we have implemented a practical website for developers looking for technology comparison knowledge. An analysis of site visit logs demonstrates the usefulness of our tool.
- We added more detailed related work analysis.

2 MINING SIMILAR TECHNOLOGY

Studies [12], [13] 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., *dijkstra, rsa*), programming languages (e.g., *golang, javascript*), libraries and



(a) Continuous skip-gram model



(b) Continuous bag-of-words model

Fig. 3. The architecture of the two word embeddings models (different arrow directions). The continuous skip-gram model predicts surrounding words given a central word, and the CBOW model predicts the central word based on the context words.

frameworks (e.g., *gson, flask*), and development tools (e.g., *sublime, vmware*). Many of them are comparable, such as (*tcp, udp*), (*nltk, opennlp*), (*swift, objective-c*), etc. 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 a similar context. Studies [14], [15], [16] show that word embeddings are better at capture rich semantic and syntactic properties of words for measuring word similarity compare with tradition n-gram model, and Chen et al.'s work [13] further confirms the effectiveness of tag embedding for inferring similar third-party libraries. 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. For example, given single tag sentences like "python, nlp, nltk, pos-tagger", the word *nltk* is encoded by its context "python, nlp, pos-tagger". Other words with similar context may thus share a similar meaning with it.

There are two kinds of widely-used word embedding methods [14], the continuous skip-gram model [17] 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:

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

while the objective of the CBOW model is:

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

Tag Wiki:	RSA	is	a	common	public	key	algorithm.
Part of Speech:	NN	VBZ	DT	JJ	JJ	JJ	NN

Fig. 4. POS tagging of the definition sentence of the tag *RSA*

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 i.e., all tags attached to the post, and each tag is a word. As a 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. The details are discussed in Section 6.1.3.

2.2 Mining Categorical Knowledge

In Stack Overflow, tags can be of different categories. 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. A TagWiki description usually starts with a short sentence to define a 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, we use NLP methods [13] to extract such nouns 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. We then 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 the small-scale pilot study show that when using Python, NLTK performs slightly better than Standford-CoreNLP and spaCy. Hence we adopt NLTK in this work. In NLTK, the noun is annotated by different POS tags [18] 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 *RSA*. Based on the POS tagging results, we extract the first noun (*algorithm* in this example) after the be verb (*is* in this example) as the category of the tag. That is, the category of *RSA* is *algorithm*. Note that if the noun is some specific word, such as *system* or *development*, we will further check its neighborhood words to see if it is e.g. *operating system* or *independent development environment*.

With this method, we obtain 318 categories for the 23,658 tags (about 67% of all the tags in TagWiki). We manually normalize these 318 categories labels, including 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 obtained 167 categories and categorized

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

them into five general categories: programming language, platform, library, API, and concept/standard [19]. 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 tag *itext* is "Library to create and manipulate PDF documents in Java", or for *markermanager*, the tag definition sentence is "A Google Maps tool". As there is no *be* verb in this sentence. According to our observation, for most of such cases, the category of the tag 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 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. Note that one tag may be corresponding to multiple category match and we take all of them into consideration. Using the dictionary look-up method, we obtain the category for 9,648 more tags.

2.3 Building a 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.,

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

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 similar technologies. 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, but they are not comparable libraries to the *nltk*. Some mis-tagging may also negatively influence the quality tag embedding [20], [21]. In our approach, we rely on the category of tags (i.e., categorical knowledge) to return only tags within the same category as candidates and also help mitigate the noise. Some examples can be seen in Table 1. In the third line of the table, the tag *vim-plugin* is defined as Library categories, which is different from *vim*. So we remove it from the *vim* similar technology list.

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 $Thresh$. We set $Thresh$ as 0.4 according to our previous work [22], and a smaller threshold leads to low accuracy whereas a larger one results in low coverage. 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. Previous works [23], [24] have defined comparative opinions as sentences that are expressed in a comparative form (e.g. "X is better than Y") where people compare two objects. In grammar constructs, people use comparative adjectives/adverbs to compare objects. We summarised comparative sentence patterns according to the comparative adjectives/adverbs and our observations of posts in Stack Overflow. Note that technology comparison may also be expressed in different forms such as one single sentence, consecutive sentences, code fragments, tables or figures. But we only take the natural-language sentences into consideration since it is the most common way of doing technology comparison. In our work, we extract the comparative sentences first. Then, we measure the similarity among the comparative sentences, and cluster them into several groups, each of which we aim will identify a prominent aspect of the technology comparison that users are concerned with.

3.1 Extracting Comparative Sentences

To extract comparative sentences, we first carry out some preprocessing to 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 into individual sentences by punctuations like ".", "!", "?". We remove all sentences ending with question marks, as we want 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.

In a technical discussion in Stack Overflow, developers may adopt some pronoun like "it", "that", "which" to represent the technology. For example, given the paragraph in Figure 5, the two "it" in the second sentence refers to "postgresql" in the first sentence. However, with some cases, the reference may be too far from its source, leading to a

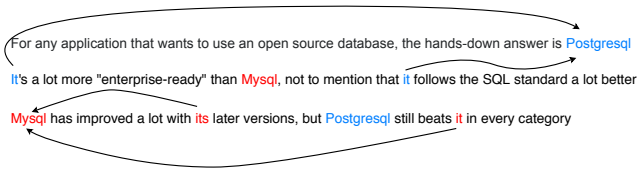


Fig. 5. An example of coreference resolution in comparative sentences.

TABLE 2
Examples of alias

Tech term	Synonyms	Abbreviation
photoshop	adobe_photoshop, photoshops	ps
dataframe	data-frame, pd.dataframe	df
libgd	gd_library	gd
microsoft sql server	ms-sql, msql, ms_sql	mssql
breadth-first search	breadth first search, breadth-first-search	bfs

negative influence on the location of comparative sentences. Therefore, we adopt a coreference resolution algorithm [25] to recover the pronoun before locating comparative sentences. Conventionally, a coreference resolution is based on a set of manual-crafted rules by analyzing dependency tree of words in the sentence. However, those rules may not be scalable, therefore, we adopt the state-of-the-art neural network based coreference method, named NeuralCoref[2]. It feeds word embedding of potential words around each mention to two neural networks with one for giving score for finding a possible antecedent, and the other giving a score for a mention having no antecedent. Given two scores, we compare them and pick the highest score to determine whether the mention has an antecedent and, if so, which word it is.

3.1.2 Locating Candidate Sentences

Sentences mentioning a pair of comparable technologies may contain a comparison opinion between them. According to our observation, the comparison may occur in either one single sentence, or in contextual sentences. Therefore, the candidate comparison sentences may be single sentences mentioning two comparable technologies, or be consecutive sentences with each containing one comparable technology.

Note that using only the tag names is not enough. As posts in Stack Overflow are informal discussions about programming-related issues, users often use aliases to refer to the same technology. Aliases of technologies can be abbreviations, synonyms and some frequent misspellings. For example, there are several different alias of "visual studio" in many forms such as "visual-studio" (synonym), "vs" (abbreviation), and "visual studion" (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 the tag names. Chen et al.'s work [26] builds a large thesaurus of morphological forms of software-specific terms, including abbreviations, synonyms and misspellings. Table 2 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 two sets of comparative sentence patterns: one for the single sentence and the other for contextual sentences. The single 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 four comparative POS tags: *CV* (comparative verbs, e.g. prefer, compare, beat), *CIN* (comparative prepositions, e.g. than, over), *CCONJ* (comparative conjunctions. e.g. whereas, while), *NW* (negation words, e.g. wouldn't, doesn't), 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 four comparative patterns for single sentences, as shown in Table 3. 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 four patterns. If so, the sentence will be selected as a single comparative sentence. Note that the rules for matching the comparative sentences are adopted following the order in Table 3.

For contextual candidate sentences, we develop three patterns for identifying comparative opinions, as shown in Table 4. The first two patterns are similar to the first two patterns for the single sentence, as sometimes developers in Stack Overflow may give their comparison in two consecutive sentences e.g., "**Postgres** has a richer set of abilities and a better optimizer. Its ability to do hash joins often makes it much faster than **MySQL** for joins." Therefore, for consecutive sentences containing comparable technologies separately, we will check if either sentence fits one of these two patterns. Note that there are four single-sentence patterns and we only take the first two of them into the contextual-sentence patterns. According to our observation, the 3^{rd} and 4^{th} single-sentence patterns are not very accurate The details are discussed in Section 6.2.

The other pattern is that one contextual sentence is the affirmation (AFF), while the other is the negation (NEG), with at least one comparable technology as the subject of one sentence. For example, the prior sentence "Cassini does not support HTTPs" is the negation sentence, and the latter sentence "However, you can use IIS to do this" is the affirmation sentence, resulting in the comparison opinion that IIS can support the feature that is not owned by Cassini. To detect such a pattern, we first create a list of negation words defined as "*NW*" in POS tag such as *couldn't, wouldn't, not.*[3]. For each pair of sentences, we check if either of them match the pattern "*TECH NW*", so that we locate the negation sentence. Then, for the other affirmation sentence, it must match any one of the following patterns:"*TECH VB NN*", "*TECH VB JJ*", "*TECH VB RB*", and "*VB TECH TO VB*".

3.2 Measuring Sentence Similarity

To measure the similarity of two comparative sentences, we adopt the Word Mover's Distance [8] which is espe-

2. https://github.com/huggingface/neuralcoref

3. The full list can be seen at https://sites.google.com/view/difftechplus

TABLE 3
Patterns of comparative single sentences

No.	Pattern	Sequence example	Original sentence
1	*TECH * VBZ * (JJR ∨ RBR)*	innodb has 30% higher ubuntu is better	InnoDB has 30% higher performance than MyISAM on average. i believe that ubuntu is better than centos.
2	*((RBR JJ) ∨ JJR) * CIN * TECH*	faster than coalesce more complex than mysql	Isnull is faster than coalesce. Triggers in postgresql have a syntax a bit more complex than mysql.
3	*CV * CIN TECH*	recommend scylla over cassandra	I would recommend scylla over cassandra.
4	*CV VBG TECH*	recommend using html5lib	I strongly recommend using html5lib instead of beautifulsoup.
5	*TECH * CCONJ * TECH*	mergesort * whereas Quicksort	Mergesort worst case complexity is $O(n \log n)$ whereas Quicksort worst case is $O(n^2)$.

TABLE 4
Patterns of comparative contextual sentences

No.	Pattern	Sequence example	Original sentence
1	*TECH * VBZ * (JJR ∨ RBR)*	triple des is generally better	Triple des is generally better but there are some known theoretical attacks. If you have a choice of cipher you might want to look at aes instead.
2	*((RBR JJ) ∨ JJR) * CIN * TECH*	faster than MySQL	Postgres has a richer set of abilities and a better optimizer. Its ability to do hash joins often makes it much faster than MySQL for joins.
3	*AFF — NEG*	cassini does not iis to do	Cassini does not support https. However you can use iis to do this.

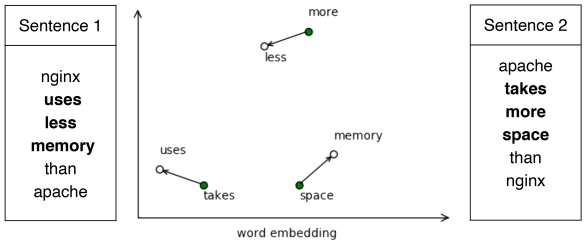


Fig. 6. An illustration of measuring similarity of two comparative sentences

cially useful for short-text comparison. In the implementation, given two sentences S_1 and S_2, we take one word i from S_1 and another j from S_2. Let their word vectors be v_i and v_j. The Euclidean distance between i and j is $c(i,j) = ||v_i - v_j||_2$. 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 from source$A_1, A_2, ...$ to target $B_1, B_2, ...$ Getting such minimum cost is a well-studied optimization problem of earth mover distance [27], [28].

To use word mover's distance in our approach, we first train a word embedding model based on the Stack Overflow post content so that we get a dense vector representation for each word. 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 $similarity\ score(S_1, S_2) = \frac{1}{1+D(S_1,S_2)}$ The higher the score is, 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[4]. We show some similar comparative sentences by word mover's distance in Table 5.

To illustrate this use of word movers' distance, we show an example in Figure 6 with two comparative sentences comparing *apache* and *nginx*: "*nginx uses less memory than apache*" and "'*Apache takes more space than nginx*'. 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 *(uses, takes)*, *(less, more)*, and *(memory, space)*. 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. 6 and Table 5 such as "But safari takes more time

4. We experimentally test the threshold as 0.45, 0.50, 0.55, 0.60, and select 0.55 as the most preferred value. Detailed results are at https://sites.google.com/view/difftechplus

TABLE 5
Examples of similar comparative sentences by Word Mover's Distance

Comparable technology pair	Comparative sentences
quicksort & *mergesort*	Quicksort is done in place and doesn't require allocating memory, unlike mergesort. Mergesort would use more space than quicksort.
swing & *awt*	Consider using swing which has much better performance over the old heavyweight awt. Yes swing has newer and better apis than awt.
google-chrome & *safari*	But safari takes more time than google-chrome browser. I get a lot of results about how safari is slower than google-chrome.
get & *post*	Post is also more secure than get because you aren t sticking". When you use post data is a a lot more safer than get and you can send large no. of request parameters.
tcp & *udp*	Tcp is a slower more reliable protocol than udp is. This is the reason why udp is much faster than tcp.

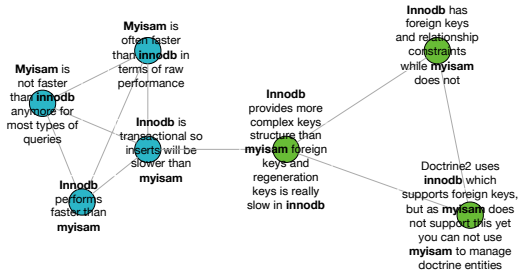


Fig. 7. Communities in the graph of comparative sentences

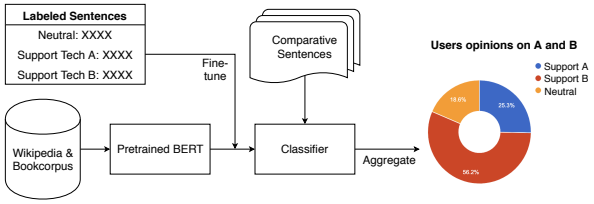


Fig. 8. Summarizing overall opinions towards each technology

than google-chrome browser" and "I get a lot of results about ho safari is slower than google-chrome"), they may still share the same comparison opinions. In graph theory, a set of highly correlated nodes is referred 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 [9], a hierarchical community detection method which has been one of the most known algorithms proposed for community detection [29]. It uses an iterative modularity maximization method to partition the network into a finite number of disjoint clusters that will be considered as communities. Given an undirected connected network, the algorithm calculates the betweenness of all existing edges first. Then it removes the edges with the highest betweenness. The two processes are repeated until all edges are removed. Then the connected nodes are classified into different communities. Note that each node must be assigned to exactly one community. Fig. 7 shows the graph of comparative sentences for the comparison of *myisam* and *innodb* (two storage engine for Mysql), in which each node is a comparative sentence, and the detected communities are visualized in the same color.

As seen in Fig. 7, 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. This 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 6 shows the comparison aspects of four communities for comparing *UDP* with *TCP*. The representative keywords directly show that the comparison between *UDP* with *TCP* mainly focuses on four aspects: speed, header, usability, and fields that they are used for.

4 Summarizing Overall Opinion

For some comparable technologies, there may be too many comparison opinions, which are too time-consuming for developers to read. To address this, we further develop a sentiment classifier based on the BERT model [10] for automatically distilling the overall opinion towards the comparable technologies. That summarization can be an important criteria for developers to determine which technology to adopt. To differentiate the pros and cons for each technology, we further break down the overall opinion into each aspect by carrying out sentiment analysis of all sentences in each cluster. Therefore, when developers are comparing comparable technologies, they can focus on the aspect that matters most. The general process of summarising overall opinion is shown in Fig. 8.

For summarizing the overall opinions toward comparable technologies, we formulate it as a classification problem, which are commonly used in the opinion extraction problems [30], [31] . Given a set of comparative sentences, we build a classifier for identifying which technology does each sentence support. By counting the total number of support sentences for each technology, we obtain the sentiment summarization of them. We replace the comparable technology pairs with two unique tokens i.e., TechA (first occurrence) and TechB (second occurrence) to generalize two technology pairs. Note that in addition to the sentence sentiment, we also need to tell sentiment direction to TechA or TechB. Therefore, we cannot use existing sentiment analysis tools [32], [33], but develop our own model.

However, a supervised learner requires a large-scale labeled dataset, which is labor-extensive and time-consuming.

TABLE 6
The representative keywords for clusters of *UDP* and *TCP*.

Representative keywords	Comparative sentences
faster slower reliable	One often finds the argument that udp is faster than tcp Udp is way lighter and faster but somewhat less reliable than tcp. Udp is generally faster than tcp as it does not have to do the overhead checking of consistency that tcp must deal with.
header connection size	You will notice that the tcp header has more fields than the udp header and many of those fields will be populated. Udp communication is connection less as compared to tcp which need a connection. The header size of udp is less than tcp.
harder, easier,travelsal	Doing p2p nat traversal over tcp is a bit harder than udp. Keep in mind that implementing udp traversal is easier than tcp. It was introduced since the nat traversal for tcp is much more complicated than udp.

To overcome this, we adopted the state-of-the-art model, BERT for this work. BERT (Bidirectional Encoder Representations) [10] is a bidirectional unsupervised language representation model that maps the words or sentences into vectors. It is designed based on the Transformer architecture with self-attention mechanism [34]. The BERT model is trained for two targets including the masked language model (i.e., predicting the words based on the context) and the next sentence prediction. These two targets make the BERT model capture both the word semantics and sentence semantics from large data, resulting in inferring high-quality semantic representation. Google released a pretrained BERT model [35] based on a large-scale corpus including Wikipedia and BookCorpus [36] which contains 2,500 million words from English Wikipedia sentences and 800 million words from 11,038 unpublished books. As the pretrained model is well trained based on such big data, fine-tuning it with small labeled dataset can still lead to good performance in down-stream NLP task like text categorization and sentiment analysis [10], [37], [38]. To leverage that learned knowledge, we only fine-tune the pretrained BERT model by adding a new layer on top of it. We freeze the parameters of existing the pretrained model but only train the final layer based on a small manually-labeled dataset for adapting it with domain-specific information.

5 Implementation

5.1 Dataset

We take the latest Stack Overflow data dump [39] (released on 4 September 2019) as the data source. It contains 18,154,493 questions with 55,665 unique tags, and 27,765,324 answers. With the approach in Section 2, we collect in total 14,876 pairs of comparable technologies. Among these technologies, we extract 26,017 comparative sentences for 2,410 pairs of comparable technologies. We use these technology pairs and comparative sentences to build a knowledge base for technology comparison.

5.2 Tool Support

Based on our proposed approach, we implemented a practical website[5] for developers [40]. With the knowledge base of comparative sentences mined from Stack Overflow, our site can return an informative and aggregated view of comparative sentences in different aspects. For example, as shown in Fig 9, given a pair of comparable technologies *Emacs* and *Vim*, the tagWiki definition is below each technology

5. https://difftech.herokuapp.com/

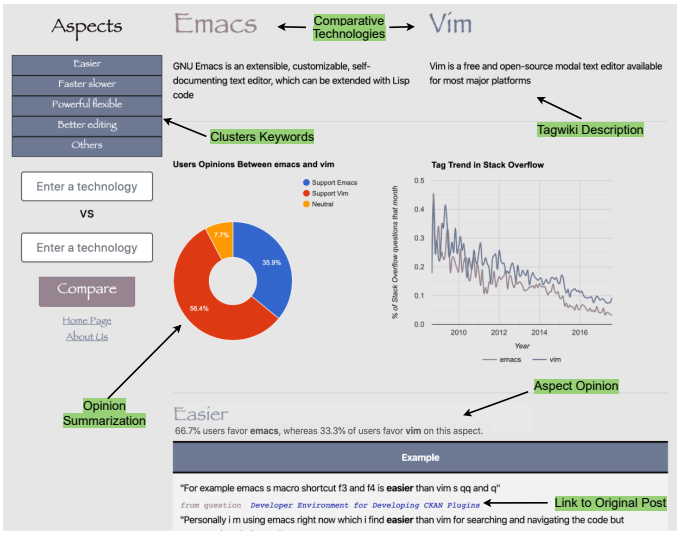


Fig. 9. The screenshot of our website DIFFTECH and some important elements in the website

name. About 56.4% of the users support using *Vim* instead of *Emacs* (shown in the doughnut chart). The post trend shows that compared with *Emacs*, there are more people talking about *Vim*. We can also see that the five clusters clarified at the top left corner. For each clustered aspect, we list the comparative sentences below and attach the direct link for each comparative sentence to its original post. Users can click the link for retrieving more content. In addition to the comparative opinions, we add the summarization of developers' preference towards the technology including the overall preference and break-down preference in each cluster. We also set up a "feedback" section in our site to receive users' feedback such as error reporting, new comparative technologies, feature requests, etc.

5.3 Visitor Analysis

We released our website and promoted it on several sites, such as StackApps [41] and Reddit [42]. We embedded Google Analytics into our website to monitor the site traffic. Fig. 10 shows the Google Analytics data from 3rd October 2019 to 5th December 2019. It shows that about 958 users from 63 different countries visited our website. These users viewed 2,234 pages and each session lasts for nearly 2 minutes. Note that most users come to our site with a very specific target, i.e., comparing a pair of similar technologies, so, the average number of visited pages in each session is rather small. Most users came from the U.S. (32.94%), with

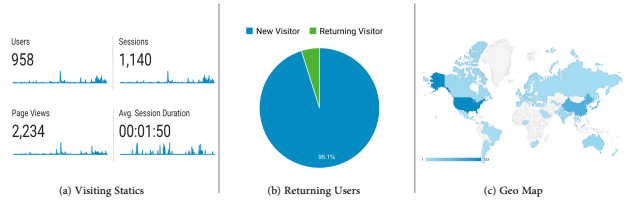


Fig. 10. The traffic of our website from Google Analytics

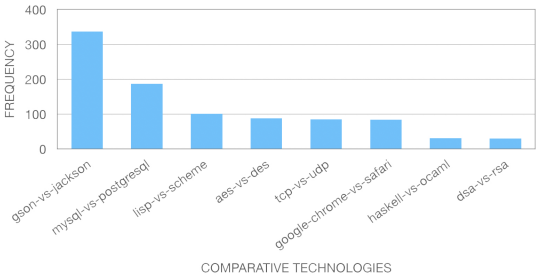


Fig. 11. Top 8 most visited pages

other countries such as China (17.51%), Japan (8.31%), etc. Nearly 5% of the users come back to visit our website again, indicating the usefulness and attraction of our site.

Among all visiting technology pairs, users compared *gson* with *jackson* most frequently (338 times) and other frequent technology pairs like *mysql vs postgresql*, *lisp vs scheme* as seen in Fig 11. Apart from the visiting, some users also post comments under our advertisements in Reddit such as "*Thank you ...will share the word*", "*It was actually better than I expected, at least for the suggested comparisons. Good job!*". They also provided some constructive suggestions for improving our tool, such as "*This sounds interesting. However I think it would be better if there were options to add to the technologies, pros, cons and usage.*".

6 Evaluation

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 recruited two Masters students to manually check the results of these three steps. Only results that they both agree upon are regarded as a ground truth for computing relevant accuracy metrics. Those results without consensus were given to the third judge, 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. For replication purposes we have released all experimental data and results on our website[6].

6.1 Accuracy of Extracting Comparable Technologies

We report on 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.

6. https://sites.google.com/view/difftechplus

6.1.1 The Accuracy of Tag Category

From 33,306 tags with tag category extracted by our method, we randomly sample 200 tags for manual checking (see Section 2.2). Note that Nassif et al [43]'s work is similar to our approach in extracting tags categories. Therefore, we take it as the baseline and also manually checked the results from it. Our tags category extraction has an accuracy of 88% whereas their accuracy is 86.5%.

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*.

6.1.2 The Importance of Tag Category

To check the importance of tag category for accurate comparable technology extraction, we set up two methods. One uses word embedding and tag category filtering, and the other uses only word embedding. The word embedding model in two methods are both skip-gram model with the word embedding dimension as 800. We randomly sampled 150 technology pairs extracted from each method, and manually checked if the extracted technology pair is actually comparable or not. Results show that the performance of the model with tag category (90.7%) is much better than that without the tag category filtering (29.3%).

6.1.3 The impact of parameters of word embedding

There are two important parameters for the word embedding, and we test 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 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 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.

6.2 Accuracy and coverage of comparative sentences

6.2.1 Coverage of comparative sentences

To demonstrate the coverage of comparative sentences of our summarized patterns, we first collect all sentences (including single or consecutive sentences) mentioning similar technologies as candidate comparative sentences form Stack

TABLE 7
The distribution of comparative sentences in patterns

Single sentence		
No.	Pattern	**Count**
1	*TECH * VBZ * JJR/RBR*	56
2	*(RBR JJ) /JJR * CIN * TECH*	58
3	*CV * CIN TECH*	17
4	*CV VBG TECH*	4
5	*TECH * CCONJ * TECH*	26
Contextual sentences		
No.	Pattern	**Count**
1	*TECH * VBZ * JJR/RBR*	15
2	*(RBR JJ) /JJR * CIN * TECH*	5
3	*AFF-NEG*	12

TABLE 8
The accuracy of comparative sentences extraction

Single sentence				
No.	Pattern	**#right**	**#wrong**	**Accuracy**
1	*TECH * VBZ * JJR/RBR*	47	3	94%
2	*(RBR JJ) /JJR * CIN * TECH*	46	4	92%
3	*CV * CIN TECH*	42	8	84%
4	*CV VBG TECH*	38	12	76%
5	*TECH * CCONJ * TECH*	43	7	86%
Contextual sentences				
No.	Pattern	**#right**	**#wrong**	**Accuracy**
1	*TECH * VBZ * JJR/RBR*	47	3	94%
2	*(RBR JJ) /JJR * CIN * TECH*	47	3	94%
3	*AFF-NEG*	45	5	90%
Total		355	45	88.8%

Overflow. We then randomly select 200 of them for the manual inspection i.e., two authors manually check if each sentence satisfies the patterns summarized by us. Table 7 shows that most (80.5%) comparative sentences are covered by our sentence patterns. Note that the total number in Table 7 is larger than the total selected comparative sentences as some sentences can be fit into multiple patterns.

We further analyse reasons why some comparative sentences are not covered by our patterns. First, due to some grammar errors or typos in the post negative influence the POS tagger, results in the unmatching to our patterns. Second, some comparative sentences are written in a casual way like "Emacs work like IDE, heavy. Vim like editor, lightweight, it basically boils down to automated conventions (Maven) vs. absolute flexibility (Ant)"

6.2.2 Accuracy of the comparative sentence patterns

We evaluate the accuracy of our approach in finding comparative sentences from the corpus. First we randomly sample 400 extracted comparative sentences (50 sentences for each comparative sentence pattern in Table 3 and Table 4). We manually check the accuracy of the sampled sentences and Table 8 shows the results. The overall accuracy of comparative sentence extraction is 88.8%, and our approach is especially accurate for the first two patterns for single sentence and all three patterns for contextual sentences. The 3^{rd} and 4^{th} patterns for a single sentence do not achieve good performance due to the relatively loose conditions. That is also why we do not use these two patterns for extracting contextual comparative sentences.

We further check the wrong extraction of comparative sentences and find that most errors from single sentence patterns are caused when the two comparative technologies are listed together for a similar feature, i.e. "*Java framework awt or swing makes more sense for something this simple*" or when the two comparative technologies are used to compare with a third technology like "*I mean it came as a surprise to me that drupal is so much faster than wordpress and joomla*". 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*".

6.3 Accuracy of clustering comparative sentences

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

6.3.1 Baseline

We set up three baselines to compare with our comparative sentence clustering method. The first baseline is the traditional TF-IDF [44] with K-means [45]. The second baseline is based on the document-to-vector deep learning model (i.e., Doc2vec [46]) with K-means. The third baseline is a BERT [10] model with K-means, and the pretrained BERT model is directly downloaded from the Google Official Site[7]. All of the methods first convert the comparative sentences into a list of vectors for each pair of comparable technologies . We then carry out K-means clustering on 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.

6.3.2 Ground Truth

As there is no ground truth for clustering comparative sentences, we manually built a small-scale ground truth. To do this, we randomly sampled 10 pairs of comparable technologies with different number of comparative sentences. For each technology pair, we read each comparative sentence and create several clusters for these comparative sentences. Note that some comparative sentences are unique without any similar comparative sentence, and we put all those sentences into one cluster. Two PhD students manually labeled the data for us. Both of them have more than 4 years of programming experience and Bachelor or Master's degree in Computer Science. They are also familiar with the 10 comparative technology pairs in their daily programming. We asked them to label the cluster categorize sentences individually at first. After the individual manual labelling, labelling, we calculated the Fleiss' kappa score [47] between them and the value ($k = 0.83$) indicates a very good agreement. They then discussed any disagreement until consensus. We take these 10 pairs as the ground truth, whose details can be seen in Table 9.

6.3.3 Evaluation Metrics

Given the ground truth clusters, many metrics have been proposed to evaluate clustering performance in the literature. In this work, we take the Adjusted Rand Index

7. https://tfhub.dev/tensorflow/bert_en_wwm_uncased_L-24_H-1024_A-16/1

TABLE 9
Ground truth for evaluating clustering results

No.	Technology pair	#comparative sentence	#cluster
1	*compiled & interpreted language*	34	4
2	*sortedlist & sorteddictionary*	18	4
3	*quicksort & heapsort*	51	5
4	*ant & maven*	83	9
5	*lxml & beautifulsoup*	52	6
6	*awt & swing*	53	7
7	*jackson & gson*	39	3
8	*jruby & mri*	25	4
9	*pypy & cpython*	72	7
10	*memmove & memcpy*	33	3

(ARI) [48], Normalized Mutual Information(NMI) [49], homogeneity, completeness, V-measure [50], and Fowlkes-Mallows Index (FMI) [51]. 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 True Positive (i.e., the number of pairs of sentences that belong to the same clusters in both the ground truth and the predicted result), FP is the number of False Positive (i.e., the number of pairs of sentences that belong to the same clusters in the ground-truth labels but not in the predicted result) and FN is the number of False Negative (i.e., the number of pairs of sentences that belongs in the same clusters in the prediction but not in the ground truth labels).

TABLE 10
Clustering performance

Method	ARI	NMI	HOM	COM	V-M	FMI
TF-IDF+Kmeans	0.09	0.24	0.35	0.28	0.26	0.25
Doc2vec+Kmeans	0.01	0.17	0.29	0.20	0.18	0.18
Bert+Kmeans	0.10	0.24	0.35	0.28	0.26	0.26
DIFFTECH	**0.65**	**0.67**	**0.76**	**0.77**	**0.72**	**0.71**

6.3.4 Overall Performance

Table 10 shows the evaluation results. The three baseline methods have similar results, whereas tf-idf and Sentence-Bert are slightly better. Our model significantly outperforms all models on all six metrics.

According to our inspection of the detailed results, we found two reasons why our model outperforms the 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. The BERT model is trained using a general corpus (like Wikipeidia), which is quite different from our technology-specific dataset, resulting in the low-quality representation of domain-specific words. Second, constructing similar sentences as a graph in our model explicitly encodes the sentence relationships. The community detection based on the graph can then very effectively put similar sentences into clusters. In contrast, for the four baselines, the error in them is accumulated and amplified by K-means in the clustering phase.

We also analysed reasons why our model make certain mistakes. First, since we cluster them based on the graph that is created by the WMD algorithm, it highly depends on the similarity score between sentences. For some very short sentence, there may be only 1 or 2 keywords which make it difficult to identify their community. Second, some comparative sentences mention multiple aspects which also make it hard to put it to any cluster.

6.4 Accuracy of Opinion Summarization

In this section, we evaluate the accuracy of our overall opinion summarization. We compare our methods with five baselines that are commonly used in sentence categorization. We implement those baselines and use four metrics to compare them with our methods, and our methods performs better in opinion summarization than them.

6.4.1 Baselines

We set up five baselines to compare with our method. The first baseline is TF-IDF [44] with SVM [52]. The second one is N-gram vectorizer with SVM. The two models firstly covert the train sentences into vectors. Then, we use the Support Vector Machines(SVM) to predict the given sentences. Another two baselines are more advanced neural network-based approaches, Convolutional Neural Network(CNN) and Long Short Term Memory(LSTM). The last baseline is FastText [53], which is a pre-trained model released by

TABLE 11
Examples of labeled sentences

Label	Sentence
Neutral	I am not sure if TechA server will be much better than TechB
Support Tech A	TechA is much better than TechB
Support Tech B	TechA has worse performance than TechB

Facebook for text classification and representation learning. To keep the comparison fair, the dataset used for training and testing our model and the baselines is the same.

6.4.2 Dataset

As we adopt the supervised model for opinion summarization, we manually create a set of labels for opinions annotation . We randomly selected 801 and 147 comparative sentences as the training and testing datasets respectively, replaced the comparable technology pairs with two unique tokens i.e., TechA and TechB (first and second occurrence) to generalize different technology pairs. For each sentence, two participants labeled it as one of three categories, including supporting TechA, supporting TechB, or neutral, shown in Table 11 The two annotators work individually and the sentence can be taken into consideration only when they reach the agreement. There are more sentences about supporting TechA or TechB, but fewer neutral sentences. To balance the data from three categories, we remove some comparative sentences about supporting certain technology, with 267 sentences for supporting TechA, 267 for supporting TechB and 267 neutral ones.

6.4.3 Metrics

As this is a typical multi-class classification task, we adopted four metrics for measuring the performance of our model including accuracy, precision, recall, and F1-score. All of these metrics are based on the four statistics: TP (true positive) represents the number of sentences that are correctly classified as one label; TN (true negative) represents the number of sentences that are corrected classified as not that label; FP (false positive) represents the number of sentences that are predicted as the label, but actually it's not; FN (false negative) represents the number of sentences that are predicted as not the label, but actually it is of the label;

Accuracy is the proportion of correct result among the whole test case: $Accuracy = \frac{TP+TN}{(TP+FP+TN+FN)}$, same to the micro F1-score when each sentence has been assigned to exactly one label.

Precision is the ratio of the correctly predicted positive records to all positive records: $Precision = \frac{TP}{(TP+FP)}$.

Recall is calculating the correctly predicted positive records among all predicted positive records: $Recall = \frac{TP}{(TP+FN)}$.

F1-score (macro) is the harmonic mean of precision and recall, which can combine both of the two metrics above: $F1 - score = 2 * \frac{Precision*Recall}{(Precision+Recall)}$.

Note that these metrics are for binary classification by default. As our task is a multi-class classification problem, we adopted the macro average [54] of these metrics for each class as the overall performance for this model. A higher value represents better performance for all the metrics.

TABLE 12
Opinion summarization performance

Method	accuracy	precision	recall	f1-score
TF-IDF+SVM	0.544	0.516	0.521	0.516
n-gram+SVM	0.748	0.733	0.739	0.733
CNN	0.653	0.621	0.627	0.62
LSTM	0.633	0.593	0.585	0.583
FastText	0.667	0.65	0.654	0.647
DIFFTECH	**0.85**	**0.842**	**0.852**	**0.842**

TABLE 13
The accuracy of comparative sentences extraction

Source	#right	#wrong	Accuracy
Super User	44	6	88%
Unix and Linux	42	8	84%

6.4.4 Overall Performance

Table 12 shows the experiment results. We can see that n-gram model with SVM has the highest among other baselines. But compared with all baselines, our model has the best performance with 13.6%, 14.8%, 15.2%, 14.8% increase than the best baseline in terms of accuracy, precision, recall and f1-score. Note that TF-IDF has the worst performance as it can't capture the semantic information of complex sentences. For deep-learning based methods, CNN, LSTM, and FastText, their performance are not as good as expected due to the small size of our training corpus. In contrast, our model is based on the pre-trained BERT model which both distills the knowledge from a very large-scale general corpus but also provides sentence-level semantic encoding.

We further checked the wrongly predicted cases. For example, our model makes a mistake in the sentence "I am not sure if VMware Server will be much better than VirtualBox." into supporting VMware. The sentence structure may be too complicated for the BERT model, but a more labeled dataset in the future may help mitigate that effect. Some other wrong predicted cases are where the sentence is comparing two technologies and focusing on the differences. For example in the sentence "char is guaranteed to be smaller than int" simply indicates that the char type takes less storage space than int type. It is a neutral expression but our model takes it to be supporting using int.

6.5 Generality of DiffTech

In order to show the generality of our method, we selected another two Q&A sites from the Stack Exchange Networks. The two sites are Super User[8], which is a site for computer enthusiasts and power users, and Unix & Linux [9], which is for users of Linux, FreeBSD and other Unix-like operating systems. We applied our approach to both sites, and collected 858 comparative sentences from the Super User site and 611 sentences from Unix & Linux. Note that the total questions of Super User and Unix & Linux is less than 3% of that in Stack Overflow. We also cluster the sentences, and summarize the overall opinions.

For the accuracy of extraction, we randomly select 50 sentences from each site and check if they are comparative sentences or not. For Super User site, the accuracy is 88%, and for Unix & Linux is 84% as seen in Table 13. In regards

8. https://superuser.com/
9. https://unix.stackexchange.com/

TABLE 14
Task Description

Task 1: emacs vs vim I'm trying to find a text editor for some coding tasks. I've heard emacs and vim are two good options. As I'm looking for a powerful, multi-language support, and easy to use tool, which one should I choose?
Task 2: apache vs nginx I'm building up a server for my website. I need it to have a better overall performance which includes handle requests fast, use less memories, and easier to set up. Which one should I use?
Task 3: virtualbox vs vmware I'm a fresh year student looking for a software that can have a virtual Linux environment for my assignment. It doesn't need too much resources, or need more functions. I just need to practice some simple command line tasks, which should I pick?
Task 4: phpunit vs simpletest I have a PHP project and want to find a proper php testing framework. I need a framework that have a better coverage and well maintained. Should I use phpunit or simpletest?

of the accuracy of clustering, our model also outperforms the other baselines. Similar to what we did in Section 6.4, we selected and labelled 50 comparative sentences as the test samples for each site. The accuracy for Super User is 0.78, and 0.76 for Unix & Linux. The detailed results of our experiments can be found on this website[10].

Note that the performance of our model in these two datasets is slightly lower than that of Stack Overflow due to several reasons. First, the users may be different, as the Stack Overflow users are mainly developers while non-developers for the Super User. As the rules of extracting comparative sentences were mainly designed by the data-driven method from Stack Overflow, the difference in content writing may influence the result. Second, for an experiment like summarizing opinions, we do not use as large a training set as Stack Overflow for fine-tuning the BERT model which leading to a degrading of performance. Despite these differences, our model still works well in general.

6.6 Usefulness Evaluation

To demonstrate the usefulness of our tool, we carried out a user study to asking participants to finish the some comparison tasks with/without DIFFTECH. We collected the quantitative and qualitative feedback from participants to verify the usefulness. For the control group (without DIFFTECH), we ask participants to use Google Search to complete the tasks as all posts in Stack Overflow has been indexed by Google and Google is more likely to be used by developers for software development related questions [55].

6.6.1 Tasks

Based on our previous software development process, we designed 4 independent tasks covering different technology categories (i.e., software, library, framework) based on our experience in learning programming and daily development tasks as shown in Table 14. Each task has a background scenario that describes the basic requirements of the task.

6.6.2 Experiment Procedure

We recruited 8 Master students who major in Computer Science with at least 2 years of programming background.

10. https://sites.google.com/view/difftechplus

TABLE 15
The comparison between our DIFFTECH and using general Google search engine with standard deviation in the bracket. * denotes $p < 0.01$, ** denotes $p < 0.05$

Tool	Avg Time (min)	Confidence	Satisfaction
Google	6.08(2.4)	3.69(1.1)	3.75(0.7)
DIFFTECH	1.85(0.8)*	4.63(0.5)*	4.69(0.6)**

All of them use Google search regularly to deal with programming issues. None of the participants were familiar with the technology comparisons in the experimental tasks.

The experiment began with an introduction to the study. Then, we explained and walked through all of the features of the DIFFTECH and participants performed a training task with DIFFTECH system to familiarize themselves with its features. After the training session, participants were asked to work on the four tasks. All of the four tasks were completed individually with no interventions or discussion by the participants. Each participant finished 2 tasks using our tool and the other 2 using Google Search. The order of tasks, and the order of using the experimental or baseline system were rotated based on the Latin Square [56], which helped to reduce learning and fatigue effects.

Participants were given a description of the tasks and up to 10 minutes to complete each task. Once, participants were asked to rate their satisfaction and confidence in the information they collected (on a 5-point Likert scale with 1 being least satisfied/confident and 5 being most satisfied/confident). At the end, participants filled in the System Usability Scale (SUS) questionnaire [57]. In addition, we conducted a semi-structured interview focusing on their perceptions and their use of features during the study.

6.6.3 Result

Table 15 shows that participants spend 1.85 minutes to finish the task with our DIFFTECH, whereas the time of using Google search is 6.08 minutes (Fig 12 shows the time distribution). Most participants agree that the aggregated comparative sentences and the summarised opinions help a lot during the tasks. Most users are more satisfied with the DIFFTECH results (87.5% of the users gave 4 or 5, 4.59 on average) than that from Google (62.5% of the users gave 4 or 5, 3.75 on average), leading to higher confidence with DIFFTECH (100% of the users gave 4 or 5 for DIFFTECH while 56.3% of the users gave 4 or 5 for Google, 4.63 vs 3.69 on average). The participants can see multiple users discuss the similar opinions from DIFFTECH, which makes them more confident about the answer, instead of scattered information in Google results. Since there are not many samples, we adopt the Wilcoxon Signed-Rank test [58] to check the significance of the differences in these metrics. It shows that our DIFFTECH is significantly better than the conventional Google Search for comparing similar technologies in terms of the time cost, results confidence and satisfaction.

We list the results from administering the System Usability Scale questionnaire to our participants in Fig 13. Most participants agree that our tool is easy to use, the functions are well integrated, and they can learn to use DIFFTECH quickly and confident in using it. They also express their desire to use this system frequently in their future technol-

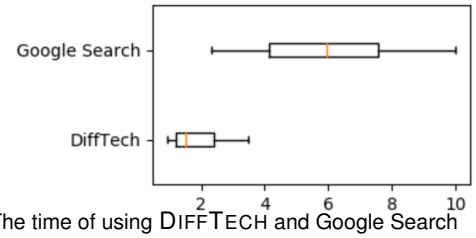


Fig. 12. The time of using DIFFTECH and Google Search

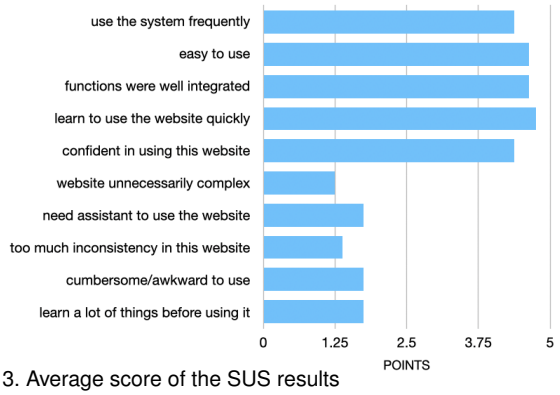


Fig. 13. Average score of the SUS results

ogy comparison tasks. Overall the participants are satisfied with the implementation of DIFFTECH.

In the post-study comments, participants said that they liked this tool as they didn't need to apply multiple searches to find the answer, and they appreciate the intuitive summarization provided by our tool. We received some feature requests such as the desire to compare more than 2 similar technologies together, or if the DIFFTECH can have some code snippets for detailed comparison. They also reported that some of the comparative technologies they would like to see are not included, but they were all happy to use the feedback function to send requests.Upon request, we will keep improving the website accordingly.

7 THREATS TO VALIDITY

There are some issues that may influence the validity of our work. We discuss the threats of validity in this section.

Internal Validity: The major threat to internal validity is the potential bias when constructing the ground truth, including tag category extraction, clustering comparative sentences, and label sentence opinions. To mitigate the threats, we have at least two participants to first independently label the same data and then resolve the disagreements by discussion. All of the participants have more than 4-year programming experience and Bachelor or Master's degree in Computer Science.

Another concern with this work is the accuracy of the tags and posts we're using from Stack Overflow. Since all content in Stack Overflow is generated by users, there may be some bias or wrong opinions in the posts or tag definitions. However, as a well-maintained most popular Q&A site, the content quality is assured by the common tagging practice and the collaborative editing mechanism [59], [60], [61]. For example, according to our empirical study of 500 randomly selected tags, the TagWiki of 72.4% of them have been revised/edited at least once since the creation. Therefore, we assume that majority of Stack Overflow data is of high quality and our data-driven approach that aggregate data pieces further mitigate that issue.

During the user evaluation, some of the participants raised that certain comparable technology pairs they would like to search for are not included on our website. This can be caused by (i) incomplete tags; (ii) errors in extraction of our approach; (iii) inaccuracies in the StackOverflow TagWiki. To mitigate this issue, we will keep improving our method in mining similar technology tags. Inspired by Nassif et al.'s work [43], we plan to include both definitions from TagWiki and tag information from Wikipedia into consideration when categorizing the technology tags. In such a way, the accuracy of mining categorical knowledge will increase and our approach shall discover more comparable technology pairs. In addition, we have added a "leave feedback" button on the website, to allow users to leave new or improved comparable pairs that they would like to know about or suggest be added. We will add them into the comparable tag list, and run our algorithm for extracting comparative opinions to them to update the website.

Some manual effort is involved during the category extraction. But note that process is only one small step of the whole pipeline, and it can be finished once offline. For most tags (77%), their categories are accurately extracted, and only a very small number of them need to be revised manually. The manual process needs to be done once and the category can be directly used in the future.

External Validity: A threat to external validity includes the generalization of our findings for more comparable technologies and for comparison of other Q&A sites. In this work, we have extracted over 14k of comparable technologies, and our experiments show that we can achieve high accuracy (88.8%) of spotting comparative sentences. As comparison opinions appear in a similar pattern, we can extract them even if more comparative technologies are obtained. In Section 6.6.1, we've showed that our model works well on other Q&A platforms like Super User and Unix & Linux. To some extent, our approach can also be generalized to other platforms with some customizations.

8 RELATED WORK

8.1 Mining similar software artefects

Finding similar software artefacts can help developers migrate from one tool to the another which is more suitable to their requirements. 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 have been adopted to mine similar artifacts ranging from high-level software [62], [63], mobile applications [64], [65], github projects [66] to low-level third-party libraries [13], [67], [68], APIs [69], [70], [71], [72], code snippets [73], queries [74] or multilingual descriptions [75], [76]. Compared with these research studies, the mined software technologies in this work have a much broader scope, including not only software-specific artefacts, but also general software concepts (e.g., algorithm, protocol), tools (e.g., IDE).

8.2 Extracting opinions about software technologies

Extracting opinions about technology preference is also important to developers, as the opinions from other developers may provide a general vision especially for novice

developers on the technology selection for their own tasks. Uddin and Khomh [77] extract API opinion sentences in different aspects to show developers' sentiment to that API. Ahasanuzzaman et al. [78] classify StackOverflow posts concerning API issues-only with Conditional Random Field (CRF) [79]. Lin et al. [80] produced a pattern-based approach that can identify overall quality, pros, and cons of APIs. Different from the works mentioned above, which focus on only extracting opinion about one certain API or library, our method tries to extract the comparison opinions of *two* similar technologies.

8.3 Comparison in Software Engineering

Given a list of similar technologies, developers may further compare and contrast them for a final selection. Some researchers have investigated such comparison, where the comparison is highly domain-specific such as software for traffic simulation [81], x86 virtualization [82], etc. Michail and Notkin [83] assess different third-party libraries by matching similar components (such as classes and functions) across similar libraries. However, these approaches can only work for comparison without the possibility to be extended to other technologies in Software Engineering. Instead, we find developers' preference of certain software technologies highly depends on others' usage experience and report of similar technology comparisons. Li et al. [84] adopt NLP methods to distill comparative user review about similar mobile Apps. De et al [85] compare the software libraries by a set of metrics. Different from their works, we adopt a light-weight approach by extracting a large pool of comparable technologies and corresponding explicit comparison in natural-language sentences, rather than heavy-weight program analysis. 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 technology comparison more easily.

Finally, it is worth mentioning some related practical projects. SimilarWeb [86] is a website that provides users engagement statistics and similar competitors for websites and applications. AlternativeTo [87] is a social software recommendation website in which users can find alternatives to a given software based on user recommendations. SimilarTech [88] 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.

9 Conclusion and Future Work

We have presented 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. We then locate comparative sentences about these technologies by coreference and POS-tag based pattern matching and organize comparative sentences into clusters for easier understanding. Finally, we summarize comparative sentences to obtain an aggregated opinion for each pair of the comparable technologies. Based on the extracted comparative opinions, we have constructed a proof-of-concept web site using it for real developers. Our evaluation shows that our system covers a large set of comparable technologies and their corresponding comparative sentences with high accuracy. Apart from Stack Overflow, our approach also successfully runs on other Q&A sites like Super User and Unix and Linux, indicating the generality of our approach.

In addition to comparative sentences explicitly mentioning both comparable technologies, some comparative opinions may also appear in code fragments, tables or figures. In the future, we will look to further improve our tool for collecting more complete comparative opinions. Therefore, we will improve our system to distill technology comparison knowledge from the current sentence level to whole-of-post levels.

Acknowledgments

John Grundy is supported by ARC Laureate Fellowship FL190100035.

References

[1] C. Chen and Z. Xing, "Mining technology landscape from stack overflow," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 14.

[2] C. Chen, Z. Xing, and L. Han, "Techland: Assisting technology landscape inquiries with insights from stack overflow," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 2016, pp. 356–366.

[3] "How to choose the right technology for your web application," https://www.langoor.com.au/how-to-choose-the-right-technology-for-your-web-application/, 2013, accessed: 2019-12-3.

[4] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou, "Extracting and analyzing time-series hci data from screen-captured task videos," *Empirical Software Engineering*, vol. 22, no. 1, pp. 134–174, 2017.

[5] "Millions of queries per second: Postgresql and mysql's peaceful battle at today's demanding workloads," https://goo.gl/RXVjkB/, 2017, accessed: 2018-04-05.

[6] "Mysql vs postgres," https://www.upguard.com/articles/postgres-vs-mysql/, 2017, accessed: 2018-04-05.

[7] C. Chen and Z. Xing, "Towards correlating search on google and asking on stack overflow," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 83–92.

[8] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *International Conference on Machine Learning*, 2015, pp. 957–966.

[9] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[11] Y. Huang, C. Chen, Z. Xing, T. Lin, and Y. Liu, "Tell them apart: distilling technology differences from crowd-scale comparison discussions." in *ASE*, 2018, pp. 214–224.

[12] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[13] C. Chen, S. Gao, and Z. Xing, "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, 2016, pp. 338–348.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] S. Ma, Z. Xing, C. Chen, C. Chen, L. Qu, and G. Li, "Easy-to-deploy api extraction by multi-level feature embedding and transfer learning," *IEEE Transactions on Software Engineering*, 2019.

[16] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S.-W. Lin, "A neural model for method name generation from functional description," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 414–421.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[18] "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, 2003, accessed: 2018-02-02.

[19] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "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, 2016, pp. 90–101.

[20] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec++: An enhanced tag recommendation system for software information sites," *Empirical Software Engineering*, vol. 23, no. 2, pp. 800–832, 2018.

[21] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 287–296.

[22] C. Chen, Z. Xing, and Y. Liu, "What's spain's paris? mining analogical libraries from q&a discussions," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1155–1194, 2019.

[23] K. D. Varathan, A. Giachanou, and F. Crestani, "Comparative opinion mining: a review," *Journal of the Association for Information Science and Technology*, vol. 68, no. 4, pp. 811–829, 2017.

[24] M. Ganapathibhotla and B. Liu, "Mining opinions in comparative sentences," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, 2008, pp. 241–248.

[25] K. Clark and C. D. Manning, "Deep reinforcement learning for mention-ranking coreference models," *arXiv preprint arXiv:1609.08667*, 2016.

[26] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, 2019.

[27] H. Ling and K. Okada, "An efficient earth mover's distance algorithm for robust histogram comparison," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 5, pp. 840–853, 2007.

[28] O. Pele and M. Werman, "Fast and robust earth mover's distances," in *Computer vision, 2009 IEEE 12th international conference on*. IEEE, 2009, pp. 460–467.

[29] Z. Jiang, J. Liu, and S. Wang, "Traveling salesman problems with pagerank distance on complex networks reveal community structure," *Physica A: Statistical Mechanics and its Applications*, vol. 463, pp. 293–302, 2016.

[30] A. Abbasi, H. Chen, and A. Salem, "Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums," *ACM Transactions on Information Systems (TOIS)*, vol. 26, no. 3, pp. 1–34, 2008.

[31] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 519–528.

[32] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018.

[33] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "SentiCR: A Customized Sentiment Analysis Tool for Code Review Interactions," in *32nd IEEE/ACM International Conference on Automated Software Engineering (NIER track)*, ser. ASE '17, 2017.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[35] "Bidirectional encoder representations from transformers (bert)," https://tfhub.dev/tensorflow/bert_en_wwm_uncased_L-24_H-1024_A-16/1, 2019, accessed: 2019-11-15.

[36] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *arXiv preprint arXiv:1506.06724*, 2015.

[37] A. Adhikari, A. Ram, R. Tang, and J. Lin, "Docbert: Bert for document classification," *arXiv preprint arXiv:1904.08398*, 2019.

[38] Y. Liu, "Fine-tune bert for extractive summarization," *arXiv preprint arXiv:1903.10318*, 2019.

[39] "Stack exchange data dump," https://archive.org/details/stackexchange, 2014, accessed: 2019-11-15.

[40] H. Wang, C. Chen, Z. Xing, and J. Grundy, "Difftech: a tool for differencing similar technologies from question-and-answer discussions," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1576–1580.

[41] "Difftech: Compare similar technologies based on stackoverflow data," https://stackapps.com/questions/8469/difftech-compare-similar-technologies-based-on-stackoverflow-data, 2019, accessed: 2019-11-18.

[42] "Website the could be helpful for programming learners," https://www.reddit.com/r/programming/comments/drba2s/website_for_developers_compare_similar/, 2019, accessed: 2019-11-18.

[43] M. Nassif, C. Treude, and M. P. Robillard, "Automatically categorizing software technologies," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[44] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[45] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[46] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[47] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[48] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[49] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, vol. 11, no. Oct, pp. 2837–2854, 2010.

[50] A. Rosenberg and J. Hirschberg, "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)*, 2007.

[51] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American statistical association*, vol. 78, no. 383, pp. 553–569, 1983.

[52] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995. [Online]. Available: https://doi.org/10.1023/A:1022627411411

[53] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[54] "sklearn.metrics.precision_recall_fscore_support," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html, 2019, accessed: 2019-11-02.

[55] M. M. Rahman, J. Barson, S. Paul, J. Kayani, F. A. Lois, S. F. Quezada, C. Parnin, K. T. Stolee, and B. Ray, "Evaluating how developers use general-purpose web-search for code retrieval," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 465–475.

[56] B. Winer, D. Brown, and K. Michels, "Statistical principles in experimental design. mcgraw-hill," *New York*, pp. 196–210, 1971.

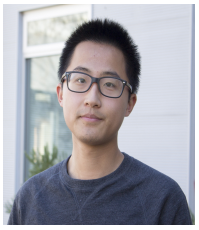
[57] P. W. Jordan, B. Thomas, I. L. McClelland, and B. Weerdmeester, *Usability evaluation in industry*. CRC Press, 1996.

[58] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

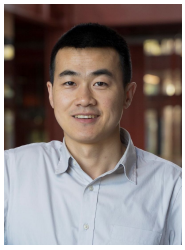
[59] G. Li, H. Zhu, T. Lu, X. Ding, and N. Gu, "Is it good to be like wikipedia? exploring the trade-offs of introducing collaborative editing model to q&a sites," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 2015, pp. 1080–1091.

[60] C. Chen, Z. Xing, and Y. Liu, "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*, vol. 1, no. 32, pp. 1–32, 2017.
[61] C. Chen, X. Chen, J. Sun, Z. Xing, and G. Li, "Data-driven proactive policy assurance of post quality in community q&a sites," *Proceedings of the ACM on human-computer interaction*, vol. 2, no. CSCW, pp. 1–22, 2018.
[62] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," pp. 364–374, 2012.
[63] F. Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 600–603.
[64] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "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, 2015, pp. 305–314.
[65] M. Linares-Vásquez, A. Holtzhauer, and D. Poshyvanyk, "On automatically detecting similar android apps," in *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.
[66] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 13–23.
[67] C. Teyton, J.-R. Falleri, and X. Blanc, "Automatic discovery of function mappings between similar libraries," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 192–201.
[68] C. Chen and Z. Xing, "Similartech: automatically recommend analogical libraries across different programming languages," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 834–839.
[69] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deepam: Migrate apis with multi-modal sequence to sequence learning," *arXiv preprint arXiv:1704.07734*, 2017.
[70] T. D. Nguyen, A. T. Nguyen, H. D. Phan, and T. N. Nguyen, "Exploring api embedding for api usages and applications," in *Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on*. IEEE, 2017, pp. 438–449.
[71] C. Chen, Z. Xing, Y. Liu, and K. L. X. Ong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, 2019.
[72] C. Chen, "Similarapi: mining analogical apis for library migration," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2020, pp. 37–40.
[73] F.-H. Su, J. Bell, G. Kaiser, and S. Sethumadhavan, "Identifying functionally similar code in complex codebases," in *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.
[74] K. Cao, C. Chen, Baltes, C. Sebastian, Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE.
[75] G. Chen, C. Chen, Z. Xing, and B. Xu, "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, 2016, pp. 744–755.
[76] X. Wang, C. Chen, and Z. Xing, "Domain-specific machine translation with recurrent neural network for software localization," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3514–3545, 2019.
[77] G. Uddin and F. Khomh, "Opiner: an opinion search and summarization engine for apis," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 978–983.
[78] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Caps: a supervised technique for classifying stack overflow posts concerning api issues," *Empirical Software Engineering*, pp. 1–40, 2019.
[79] C. Sutton, A. McCallum *et al.*, "An introduction to conditional random fields," *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.
[80] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 548–559.
[81] S. L. Jones, A. J. Sullivan, N. Cheekoti, M. D. Anderson, and D. Malave, "Traffic simulation software comparison study," *UTCA report*, vol. 2217, 2004.
[82] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 5, pp. 2–13, 2006.
[83] A. Michail and D. Notkin, "Assessing software libraries by browsing similar classes, functions and relationships," in *Proceedings of the 21st international conference on Software engineering*. ACM, 1999, pp. 463–472.
[84] Y. Li, B. Jia, Y. Guo, and X. Chen, "Mining user reviews for mobile app comparisons," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, p. 75, 2017.
[85] F. L. de la Mora and S. Nadi, "Which library should i use?: a metric-based comparison of software libraries," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, 2018, pp. 37–40.
[86] "Similarweb," https://www.similarweb.com/, 2018, accessed: 2018-04-05.
[87] "Alternativeto - crowdsourced software recommendations," https://alternativeto.net/, 2018, accessed: 2018-04-05.
[88] "Similartech: Find alternative libraries across languages," https://graphofknowledge.appspot.com/similartech/, 2018, accessed: 2018-04-05.



Han Wang received the B.S. degree with honours in Software Engineer from Australian National University, Australia, in 2017. He is currently a PhD student from Faculty of Information Technology, Monash University, Australia and is a student of Chunyang Chen. He focuses on applying techniques such as deep learning, NLP, etc., to solve software engineering problems.



Chunyang Chen is a lecturer (Assistant Professor) in Faculty of Information Technology, Monash University, Australia. His research focuses on software engineering, deep learning and human-computer interaction. He has published over 40 papers in referred journals or conferences. He is a member of IEEE and ACM. He has received ACM SIGSOFT Distinguished Paper Award in ICSE 2020, Facebook Research Award in Probability and Programming 2020, etc. https://chunyang-chen.github.io/

Zhenchang Xing is the senior lecturer at the research school of computer science, Australian National University, Australia. Dr.Xing's research interests include software engineering and human-computer interaction. His work combines software analytic, behavioral research methods, data mining techniques, and interaction design to understand how developers work, and then build recommendation or exploratory search systems for the timely or serendipitous discovery of the needed information.

John Grundy is Australian Laureate Fellow and Professor of Software Engineering at Monash University, Australia. He has published widely in automated software engineering, domain specific visual languages, model-driven engineering, software architecture, and empirical software engineering, among many other areas. He is Fellow of Automated Software Engineering and Fellow of Engineers Australia.