Improving API Knowledge Discovery with ML: A Case Study of Comparable API Methods

Daye Nam, Brad Myers, Bogdan Vasilescu, and Vincent Hellendoorn
Carnegie Mellon University, USA
{dayen, bam, bogdanv, vhellend}@andrew.cmu.edu

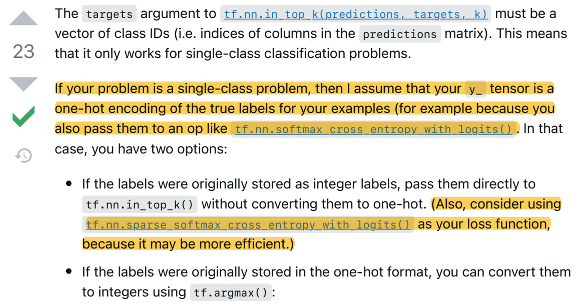
***Abstract*—Developers constantly learn new APIs, but often lack necessary information from documentation, resorting instead to popular question-and-answer platforms such as Stack Overflow. In this paper, we investigate how to use recent machine-learning-based knowledge extraction techniques to automatically identify pairs of comparable API methods and the sentences describing the comparison from Stack Overflow answers. We first built a prototype that can be stocked with a dataset of comparable API methods and provides tool-tips to users in search results and in API documentation. We conducted a user study with this tool based on a dataset of TensorFlow comparable API methods spanning 198 hand-annotated facts from Stack Overflow posts. This study confirmed that providing comparable API methods can be useful for helping developers understand the design space of APIs: developers using our tool were significantly more aware of the comparable API methods and better understood the differences between them. We then created SOREL, an comparable API methods knowledge extraction tool trained on our hand-annotated corpus, which achieves a 71% precision and 55% recall at discovering our manually extracted facts and discovers 433 pairs of comparable API methods from thousands of unseen Stack Overflow posts. This work highlights the merit of jointly studying programming assistance tools and constructing machine learning techniques to power them.**

I. Introduction

New libraries and frameworks constantly emerge, each with their own Application Programming Interfaces (APIs), so developers must frequently learn new APIs to stay competitive. Indeed, researchers have concluded that "being able to learn new technical skills is likely more important [to a software engineer] than any individual technical skills" [1].

Learning a new API requires many types of knowledge [2] and there are many documented challenges to obtaining these [3]. Among them, two key challenges are: finding the appropriate API types and methods needed for a particular task (i.e., *discoverability* [3]–[5]), and identifying all the relevant information about their execution behavior, usage patterns, side effects, alternatives, *etc.* [6]–[10].

Lacking such knowledge can affect developers in at least two important ways. First, by its very nature [11], software engineering work involves careful consideration of design alternatives and often tradeoffs between competing objectives. This happens at all levels, including when choosing, within a given library or framework, the most appropriate API types and methods to use while balancing complexity, runtime performance, memory consumption, but also readability, understandability, amount of boilerplate involved, *etc.* , as there are typically multiple ways to implement the same functionality [12]. Thus, one can expect that the harder it is for developers to access the relevant information to make informed decisions, the less appropriate their choices would be. Second, although modern APIs are often large and sophisticated, many of their types and methods are rarely used in practice, in part due to insufficient discoverability [13]. In addition to affecting the aforementioned decisions, this can also slow down software developers, and, in the long term, lead to missed opportunities for developers to take advantage of the most effective technological solutions.


23

The `targets` argument to `tf.nn.in_top_k(predictions, targets, k)` must be a vector of class IDs (i.e. indices of columns in the `predictions` matrix). This means that it only works for single-class classification problems.

If your problem is a single-class problem, then I assume that your `y_` tensor is a one-hot encoding of the true labels for your examples (for example because you also pass them to an op like `tf.nn.softmax_cross_entropy_with_logits()`. In that case, you have two options:

- If the labels were originally stored as integer labels, pass them directly to `tf.nn.in_top_k()` without converting them to one-hot. (Also, consider using `tf.nn.sparse_softmax_cross_entropy_with_logits()` as your loss function, because it may be more efficient.)
- If the labels were originally stored in the one-hot format, you can convert them to integers using `tf.argmax()`:

Fig. 1: Fragment from a Stack Overflow answer by mrry / CC BY-SA 3.0 illustrating the tacit crowd knowledge on comparable API methods `softmax_cross_entropy_with_logits` and `sparse_softmax_cross_entropy_with_logits`. We highlighted the sentences supporting the comparison.

In acquiring API knowledge, developers tend to consult a variety of sources, as no single one is typically complete. These include the official reference documentation [14], but also crowd-based knowledge sharing websites like Stack Overflow (SO), which have long been recognized as usefully complementary [15]–[20]. In particular, Stack Overflow discussions are a good source of knowledge on API method comparisons, which developers ask about quite frequently [6]. Such comparisons can expose developers to more diverse ways to use an API, thereby improving discoverability, and can also surface more relevant execution facts about the API, thus improving learning and programming using the API. Yet, this information on comparable API methods is generally not easy to extract and not available on demand, despite its value.

Sometimes, Stack Overflow discussions start with the author asking explicitly for a comparison between API methods, e.g., "What is the main difference between `StringBuffer` and StringBuilder?" Prior research using natural language processing (NLP) techniques can identify such posts with high accuracy when the language is well structured and amenable to detection using syntactic patterns [6], [21], [22]. However, comparisons between API methods on Stack Overflow are also often tacit—they arise organically as part of answers, without the original poster asking for them explicitly, and the relevant sentences for the comparison are not clearly identifiable or even grouped together, but rather can be scattered throughout a post. Consider the example in Figure 1, where the author of the answer offers relevant information about the relationship between two methods from the popular machine learning TensorFlow API, softmax_cross_entropy_with_logits and sparse_softmax_cross_entropy_with_logits, unprompted by the question. It is sometimes difficult even for humans to recognize that the answer compares the two methods, and then to extract the most relevant sentences informing the comparison (indeed, we ourselves had difficulty with this; see Section III). Therefore, it is clearly challenging to design a pattern-matching-based classifier to do the same, since in addition to recognizing which answers include such API method comparisons, one would also need to extract and encode generalizable syntactic patterns for training.

Instead, in this paper we show that (semi-)supervised, deep-learning based methods recently developed in the NLP knowledge extraction community [23] can learn to recognize API method comparisons *directly from labeled data*, bypassing the need to manually identify elaborate syntactic patterns. Moreover, we show that such models can be trained efficiently, with reasonable effort for data labeling, thus taking the next step towards the goal of automatically extracting knowledge from unstructured Stack Overflow posts.

Our work consists of three main parts. First, we develop an **annotation protocol** and use it to compile a **dataset** of 266 pairs of comparable API methods identified in a statistically representative sample of Stack Overflow answer posts discussing the TensorFlow API. As part of our annotation effort, we also label the sentences within a post that are most relevant in support of the comparison.

Second, we design and run a **human subjects study with 16 participants completing a series of tasks in two experimental conditions**, *viz.,* with and without access to a custom-built browser plugin that augments the TensorFlow API reference documentation with the information on comparable API methods from our labeled dataset. We analyze both quantitative and qualitative data from the human study, showing, among other things, the extent to which having access to such information can help with understanding the API design space given task requirements, and what requirements participants have for such a tool.

Third, informed by our annotation effort and human subjects study, we develop and evaluate **SOREL** (Stack Overflow RELation extractor), **a deep-learning-based knowledge extraction engine**. SOREL identifies pairs of comparable API methods *and* explanations of their relationship from unseen Stack Overflow answers by learning and extrapolating from our hand-annotated dataset. Our evaluation results show that SOREL outperforms baselines and pattern-matching-based approaches, and can discover relevant novel facts compared to the training data, the official documentation, and simple Google search. As such, the SOREL approach, which could be applied analogously to other APIs beside TensorFlow as long as they are covered by sufficiently many Stack Overflow discussions, can be useful to API designers and tool builders looking to improve API documentation and learning.

II. Related Work

Crowd-based knowledge in general, and question & answer websites like Stack Overflow in particular, have long been an indispensable resource to software developers and the focus of much research attention. There is a rich literature studying Stack Overflow to understand what challenges developers face in practice, e.g., [6], [17], [19], [24], [25], helping to curate and navigate the relatively unstructured Stack Overflow content, e.g., [8], [26]–[31], and extracting knowledge from Stack Overflow to augment more traditional forms of documentation, e.g., [9], [15], [16], [18], [20]; researchers have also been developing tools to more closely integrate such knowledge into the development workflow, e.g., [18], [32]–[34].

In this context, our work relates to augmenting API documentation with knowledge extracted from Stack Overflow. Many types of knowledge have been in focus, including common use and misuse patterns [8], [9], [34], [35], caveats [16], [36], [37], opinions on different quality attributes (e.g., usability) [29], [38]–[41], or more generally any Stack Overflow posts discussing some given API methods, such as those invoked in the developer's local IDE context [42]. Separately, there have also been efforts to automatically summarize Stack Overflow posts [27]–[29], [38], [43]–[46] to make it easier for readers to find relevant information.

At the intersection of the previous two directions and closest to our work, there have been a few notable efforts to extract the most relevant individual sentences containing particular insights about a given API type [16], and even to identify comparable technologies, including libraries and API methods, and extract the relevant supporting sentences for the comparison [21], [22], [47]. What all these approaches have in common is a need to define syntactic patterns (heuristics) both for the relation extraction sub-task (i.e., finding the comparable API methods) as well as the supporting sentence extraction sub-task (i.e., finding the most relevant sentences for the comparison). While they have all been shown to be accurate on their respective benchmarks, fundamentally, approaches like these have inherently limited recall because of the manual annotation and analytical effort required to discover all the relevant and generalizable linguistic patterns.

In contrast, our work shows that comparable API methods and the supporting evidence for the comparison can be identified with similar or higher accuracy *directly from examples* (see Section VI-D for the evaluation results comparing our approach to the relevant prior work baselines). Our key idea is to build our approach not on linguistic patterns but rather on using (semi-)supervised learning methods, the other class of techniques to extract relational facts from raw text [48].[1] Currently these models are typically deep neural networks, trained to classify the presence or absence (and type) of a relation between two entities in a document [51], as well to predict the (smallest set of) sentence(s) or phrases that support the validity of an extracted relation. Most state-of-the-art relation extraction tools use such deep learners, often achieving high precision given sufficient training data. Driven by this success, recent research has expanded the application of these methods, both in terms of scale [52] (from sentence to document-level relational extraction) as well as the type of features extracted, such as learning to differentiate between (many) types of entities and relations [23], [53]. Compared to pattern-based methods, neural models tend to need substantially more labeled training data, which makes it harder to apply them to new environments such as ours. As a solution, researchers commonly use models that were pre-trained on large, existing datasets [54], [55], and fine-tune them with with relatively few training samples [56], [57] for new domains. In this work, we fine-tune BERT [58], a large pre-trained model, for comparable API methods extraction and show the effectiveness of this strategy through the evaluation of SOREL.

Our work can also be viewed as related to the API migration literature [10], [59]–[64]. Typically, such research facilitates the process of migrating code from one language to another (e.g., Java to C# [59]), one library or framework to another (e.g., PyTorch to TensorFlow [64]), or one version of an API to another (e.g., TensorFlow 1.x to TensorFlow 2 [65]). As such, the typical goal is to find (and, if possible, automatically migrate to) the semantically equivalent implementation of some source code into a target language, library, framework, or version. Our goal in this paper is broader in some sense and narrower in others—broader in that we are interested in *comparable* but not necessarily semantically equivalent API methods, together with an explanation on how they are related; and narrower in that we only evaluate our technique on API methods from the same library, rather than across libraries or languages. Also, the main motivation is different—to facilitate API discovery and API learning in our case, i.e., to help developers make informed choices about API methods, versus to migrate code automatically. Still, there is some conceptual overlap between our work and some of the API migration literature, e.g., [63], [66]–[68], most clearly when it comes to using contextual information to learn representations (embeddings) for API elements and computing the distance between them to identify pairs of comparable / analogous API methods.

[1]Relation extraction (RE) is a popular research topic in NLP focused on extracting relational facts between *entities*, typically from plain text documents. For example, in the sentence "*Rafael Nadal* is a Spanish *professional tennis player*," we may identify two entities (italicized) connected via an Occupation relation. Sentences may have more than one such relation, or entities that are not used in any. It is generally important that the text support the relation. The types of facts derived through RE can be used to create large-scale ontologies or knowledge graphs [49] based on large volumes of text (e.g., all of Wikipedia), question answering [50], and many other tasks.

TABLE I: Summary statistics for our annotated data.

Variable	Count
Annotated SO TensorFlow answers	587
SO TensorFlow answers with comparable API methods	198
Identified comparable TensorFlow API pairs	266
Unique TensorFlow methods mentioned in the annotated answers	642
Unique TensorFlow methods with comparable API methods	279
Sentences in the answers	4,298
Supporting sentences for the comparable API methods	737

III. A Benchmark of Comparable API Methods

Before we test the benefits of providing comparable API methods (Section IV) and automate the extraction (Section VI), we first create a benchmark of pairs of comparable API methods, including supporting sentences from Stack Overflow. In particular, we focus on the popular TensorFlow machine learning package (45,996 questions; 33,460 answers with at 1+ votes). However, our approach is not fundamentally limited to TensorFlow and could be applied analogously to any other API with available crowd-based knowledge.

Preprocessing. We started from the 33,460 TensorFlow-tagged answers with a score of 1 or higher returned by the Stack Exchange Data Explorer[2] as a pool of answers from which to extract comparable API methods. To increase the chance of finding comparison relations, we then selected only the 2,014 answers containing at least two TensorFlow API methods, as defined in the official list of 6,755 TensorFlow symbols.[3] Next, we randomly sampled 600 answers (98% confidence level with 4% margin of error) from this set for manual analysis. Table I lists basic statistics for our sample, after filtering out 13 of the 600 answers ($\sim$ 2%) that were longer than 512 words, i.e., more than our deep-learning model can handle efficiently.[4]

Annotation Protocol. Next we developed a labeling protocol to extract, for every Stack Overflow answer in our sample, a) the pairs of comparable API methods mentioned in the answer; b) for every pair, a list of the most relevant sentences from the answer, describing how the methods are related. For example, given the Stack Overflow answer in Figure 1, we annotate:

- API_1: `softmax_cross_entropy_with_logits`,
- API_2: `sparse_softmax_cross_entropy_with_logits`,
- Supporting evidence / summary: [sent_2, sent_5].[5]

As discussed in Section I, creating such annotations is both time-consuming and difficult: relations manifest in a diversity of ways and are often not explicit or well structured, but inconsistencies in labeling risk wreaking havoc on any downstream learner when using so little data. To ensure that our annotations were consistent, replicable, and generalizable, we created a detailed annotation protocol, refined through several pilot phases. The final protocol (see our replication

[2]https://data.stackexchange.com/ accessed Oct 19, 2020.
[3]https://www.tensorflow.org/api_docs/python/tf/all_symbols
[4]Only 3 of the 13 (0.5% of the sample) contained comparable API methods.
[5]Our sentence indexes start with 0.

package) contains annotation steps, the definition of the relation, examples, and notes about edge cases.

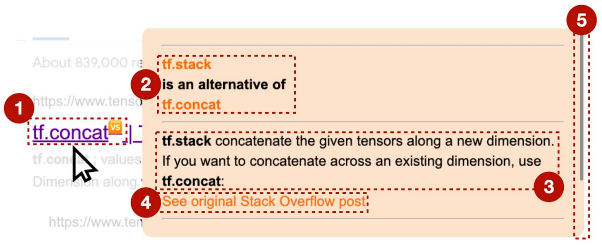
To increase validity, two authors annotated each set of documents separately, measured their inter-annotator agreement (IAA), discussed disagreements, and updated the instructions. We used the free-response kappa IAA metric [69], as the more traditional Cohen's Kappa requires all observations to be enumerated, whereas in our case only positive instances (i.e., relations between two entities and supporting evidence for each relation) are noted, not negative ones (e.g., pairs of entities not having any relation and sentences not supporting relations). Arguably, all pairs of entities without relations could be considered as negatives, but this number is often vast, growing quadratically with the number of terms (themselves not truly bounded, as annotators are allowed to add new entities), and would thus yield near-perfect agreements almost by definition. Free-response kappa does not depend on the number of negative instances. The IAA score for the first round of separate annotations (23 documents) was 0.30, and it rose to 0.82 in the second round (38 documents), after resolving disagreements and refining the protocol. As values over 0.8 are generally regarded as good agreement, both by the free-response kappa proponents [69] and the interpretation guideline for Cohen's kappa [70], we considered the instructions adequate at that point. After this, a single author annotated the remaining documents alone following the final instructions.

Resulting Dataset. Table I summarizes basic statistics for the resulting dataset: the 198 answers with comparable API methods contained around three TensorFlow methods each; around one tenth of all TensorFlow symbols were mentioned as part of any comparison; around one-third of posts contained (typically several) related pairs, which are best described with ca. three sentences on average.

IV. User Study

Next we conducted an IRB-approved human subjects study to investigate how providing developers with (1) a list of comparable API methods and (2) for each pair, a textual description of the comparison can assist developers in using new APIs more effectively. We hypothesize that such information will help developers not only **discover more of an API**, but also **understand the API better**, such that they can make more informed decisions about which methods are applicable to their task or which are preferable given alternatives.

To test these hypotheses, we first built a Chrome browser extension (Figure 2) that displays the information on comparable API methods we collected during our previous annotation effort. We deliberately chose a tooltip design to reduce information overload (tooltips only show contents when users trigger them), and to make the crowd-based information easily distinguishable from the official documentation (Stack Overflow posts can more easily become outdated [8], [71]). The tooltip is available on any webpage, including the official API documentation, Google search result pages, and Stack Overflow. We then ran a study that involves participants completing a set of tasks in two conditions, with and without

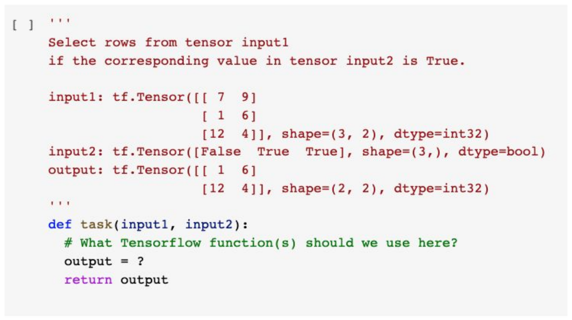


tf.stack
is an alternative of
tf.concat

tf.stack concatenate the given tensors along a new dimension. If you want to concatenate across an existing dimension, use **tf.concat**:

See original Stack Overflow post

Fig. 2: Overview of our browser plugin: (1) When comparable API methods exist in our labeled dataset, the extension inserts a "vs" icon. The user can hover over it to activate the scrollable tooltip, which displays (2) the pair(s) of comparable API methods, each with links to their reference pages; (3) the relevant sentences for the comparison; (4) a link to the Stack Overflow answer where the sentences were extracted from.



```
[ ] '''
    Select rows from tensor input1
    if the corresponding value in tensor input2 is True.

    input1: tf.Tensor([[ 7  9]
                       [ 1  6]
                       [12  4]], shape=(3, 2), dtype=int32)
    input2: tf.Tensor([False  True  True], shape=(3,), dtype=bool)
    output: tf.Tensor([[ 1  6]
                       [12  4]], shape=(2, 2), dtype=int32)
    '''
    def task(input1, input2):
      # What Tensorflow function(s) should we use here?
      output = ?
      return output
```

Fig. 3: Example of a task presented to participants.

access to the tooltip extension, and collected both quantitative and qualitative data on task performance and tool use.

A. *Study Design*

1) Participants: After advertising our study broadly inside the university community (Slack channels, posted flyers in Computer Science buildings, and personal contacts), we recruited 12 participants (6 men, 6 women) having a general understanding of machine learning (ML), who could understand the task requirements (9 PhD and 3 MS students, all in ML-relevant fields). We further recruited 4 participants (all men, 1 research scientist, 1 ML lead, 2 MS students) from outside our university after advertising our study on Twitter. To determine if participants were eligible, we administered a screening survey to collect their self-reported level of ML expertise and their main ML framework. To minimize the possibility of the participants knowing solutions to our tasks without needing to search, we specifically looked for participants who had not used TensorFlow for more than 6 months. We compensated each participant with a $15 Amazon Gift card.

2) Tasks: We designed a diverse set of eight ML-related programming tasks that mimic real-world TensorFlow use, ranging from tensor manipulation to image processing. For each task, participants were given the requirements as a short natural language description, an example input–output pair, and some starter code, and were asked to complete the implementation using appropriate TensorFlow API methods (see Figure 3 for an example and our replication package for the complete list). We intentionally designed the tasks to have more than one acceptable solution (involving different TensorFlow API methods), so we could better test the participants' understanding of all possible options.

3) Experimental Design: We chose a within-subjects design, where each participant was assigned four of the possible eight tasks (to keep the participation effort manageable), two with our browser plugin enabled (treatment) and two with it disabled (control). We used the Youden square [72] (incomplete Latin square) procedure to counterbalance the tasks and order in which they are presented to participants, to prevent carryover effects. Control (plugin disabled) and treatment (plugin enabled) were randomly assigned. Overall, each of the possible eight tasks was used four times in the treatment condition and four times in the control condition.

4) Procedure: We conducted the study via a video conferencing tool, with each session taking about 60 minutes. At the beginning of the study, we asked participants to install the browser plugin, share their screen, and think aloud while completing the tasks. Before their first task in the treatment condition, we introduced the plugin briefly and showed the participants a short demo of how it worked. We also informed participants that there could be multiple solutions to a task, and asked them to make deliberate choices. Participants were free to use or read any web pages. The experimenter logged their web search queries and the pages they visited.

To complete each task, we asked participants for their chosen API method names (but not to run any code). We then asked a few interview questions to understand their prior knowledge with the task and whether they understood the different options to make an informed decision, i.e., to list the API methods they considered and briefly describe the differences between them. At the end of the study we conducted a general interview eliciting participants' impressions of using the plugin and the usefulness of having the information on comparable API methods in completing the tasks. See our replication package for interview protocols.

B. Analysis

1) Data Collection: For qualitative data, we transcribed the interview parts of the video recordings. For quantitative data, we computed six outcome variables: (1) task completion **time** (in seconds); (2) number of search **queries**; (3) number of web **pages** visited; (4) **correctness** of the participant's solution; (5) the participant's **awareness** of comparable API methods in that context; and (6) their **understanding** of the differences between the comparable API methods. We considered a participant's answer to be correct if their solution matched our ideal solution, which we determined based on our expert knowledge of the tasks. Note that the tasks were purposefully designed to have multiple potential solutions, with varying implications and trade-offs, so "incorrect" solutions may not necessarily be "bad" solutions. We discuss this further in Section IV-C. Awareness and understanding were measured by labeling the post-task interviews, where we asked participants to discuss the difference between the multiple solutions. We rated participants' awareness as 0 if the participant answered that they did not know other relevant API methods or admitted that they guessed the difference, and 1 if they could list comparable methods in the given context. Understanding was rated as 0 if the participant appeared not to have a clear understanding of the differences or guessed based on method names, and 1 if they could describe at least one valid difference between the two methods. To account for a possible confounding factor we also rated each participant's **prior knowledge** of the task based on the interview responses, on a scale ranging from 0 (has no experience) to 3 (recalls the method name without search).

2) Analysis: To compare the six outcomes between tasks completed in the treatment and control conditions we estimated six mixed-effects multivariate regression models (one per outcome variables), with **prior knowledge** and the **condition** (treatment vs control) as fixed effects, and random intercepts for **task** and **participant** to account for variation in task difficulty and participant ability. See Appendix for summaries of the estimated regressions.

For the qualitative analysis, we first annotated the transcripts with the participant events (e.g., "opened [page url]"), and two authors open coded two transcripts separately [73], then met to discuss and refine the code book. The first author then coded all remaining transcripts, synthesized the codes, and identified emerging themes. We coded selectively on how participants interacted with the tooltip, whether and how they used the information from the tooltip as part of their answers to the tasks (especially the summary sentences extracted from Stack Overflow answers), how they used web search outside of using the tooltip, characteristics of the instances when the information provided by the tooltip seemed less relevant, general impressions on the value of having access to information on comparable API methods, and requirements for search assistance tools such as our browser plugin.

C. Results

None of our models for **time**, **queries**, or **pages** shows a statistically significant effect for **condition** at $\alpha = 0.05$. Thus, **we do not find sufficient quantitative evidence to conclude that having access to information on comparable API methods has a significant impact on task completion times or web search queries.** However, it is possible that our tasks were insufficiently complex[6] to uncover such differences between conditions with statistical confidence.

We also do not find statistical evidence that presenting comparable API methods assists developers in selecting what we consider as the ideal API methods for a given task: Although participants in the treatment condition solved the task correctly slightly more often than in the control (average score of 0.75 vs 0.66), the **correctness** model does not show a statistically significant effect for **condition** at $\alpha = 0.05$.

[6]E.g., on average users made 1.3 (treatment) to 1.4 web searches per task.

However, we reiterate that our measure of correctness is based on our expert-determined "ideal solutions" given the efficiency and design rationale of the API methods, which may not align with the solutions chosen by participants. From the interviews, we later discovered that some participants did in fact choose incorrect solutions (that still met the task requirements) on purpose, citing factors such as readability or familiarity, even though these solutions may have been less efficient or more verbose, e.g., "I chose not to do it because it seems like it's a little more complicated, it's less readable" (P13).

We did find clear evidence of increased **awareness** (coefficient = 3.03, z(59) = 3.18, p = 0.0015) and increased **understanding** (coefficient = 2.64, z(64) = 2.77, p = 0.0057) of the API methods in the treatment (tooltip) condition, supporting our hypothesis that **presenting comparable information helps developers understand the design space of API**: the odds of being aware of comparable API methods are about 20 times higher ($\exp(3.03)$) among participants with access to the tooltip information compared to those without; similarly, the odds of understanding the differences between the comparable API methods are about 14 times higher. Many participants typically discovered the API methods they ended up submitting as their answers from among the comparable API methods suggested by the tooltip in the treatment condition. Specifically, we found that in more than half of the tasks (17 participant-task pairs out of 32), participants *newly* discovered the API methods *they submitted as their answers*, among the comparable API methods suggested by the tool. In the remaining cases, participants retrieved their solution from the Google search results directly without using the tool at all (2 out of 32), focused more on the Google search results than what the tool presented (4 out of 32), or forgot to use the tool (3 out of 32) as they felt the pressure to finish the tasks on time. Finally, there were participants who did nearly no searching (6 out of 32) due to their prior experience with other libraries (e.g., PyTorch). We conclude that information on comparable API methods can help developers discover and identify which API methods to use.

From the qualitative analysis one common theme was also that **the tooltip information on comparable API methods was often novel and welcome**. Almost all interviews appreciated having the list of comparable API methods, both for discoverability reasons (e.g., "*[otherwise] it would have been pretty hard for me to have actually found the correct documentation.*"–P16) as well as usability reasons (e.g., "*The tool allowed me to explore more methods more easily in the same page without retyping the search keyword.*"–P1). A few participants mentioned that such a tool could help close the "lexical gap" between search queries and web documents, e.g., "*Sometimes, I'm not-so-clear about what I'm looking for*" (P12).

We also found that **the information on comparable API methods was often less accessible outside the tooltip**, i.e., the comparable API methods identified from Stack Overflow were sometimes not mentioned at all among the top web search results, and the official API documentation typically did not have "functional" links between the relevant API methods. A notable exception was that the reference documentation page's navigation bar lists all TensorFlow methods in alphabetical order, and the relevant comparable API methods for one task were close together in this list since they had similar names, enabling a participant to identify the second method easily.

Many participants expressed that **displaying less relevant comparable API methods did not reduce the usefulness of the tool:** out of 14 participants who discussed false positives during the interview, 9 expressed that they are okay with having some less relevant API methods in the tool results if there are also good ones among them, whereas only 3 mentioned that false positives hindered their ability to fully use the tool. Participants mentioned that going through the comparable API methods was still useful even when they are less relevant to their tasks because "*this list is still smaller [than the number of pages they need to check in general]. It is in some sense narrowing down the search, which is helpful*" (P11). P6 even took them as a benefit, saying that "*If we got exposed to potential comparable API methods out there [during search], there will be more chances to explore our uses in methods or contents in the package [in future uses].*" Other participants mentioned that the tool was less useful when too many comparable API methods are shown, but interviewees typically appreciated the tooltip design "*because it's really not intrusive*" (P5).

The summary sentences displayed below each pair of comparable API methods were useful when considered, but were used by fewer participants. When considered, the summary sentences seemed to be the way participants learned about the differences between the comparable API methods. These participants did not typically make additional web search queries to understand the differences after reading the summaries, and were able to clearly explain the rationale behind their choice of API method in the interview. As one interviewee put it, "*I think [the summary] was abstracted quite well. So even though I didn't go inside the Stack Overflow pages to see the detailed comparison, I could understand what's the gist of the differences.*" (P6).

Among participants who saw less use for the summaries, some indicated they prefer "*looking for code snippets [without] reading the text*" (P10), others that they prefer to have all the context, not just the summary (e.g., "*I found it a lot nicer to click towards the original documentation.*"–P16), and still others expressed skepticism towards the quality of crowd-based information (e.g., "*I don't really trust the text in the tools much because maybe it came from Stack Overflow.*"–P13).

D. Threats to Validity

We only studied tasks in machine learning, requiring TensorFlow API methods, which may not be representative of other API learning scenarios. Furthermore, our tasks don't address all aspects of ML programming; topics related to scalability (e.g., loading a large dataset) and performance (e.g., optimizing memory usage) are notably missing. The value of providing comparable API methods may differ in more complex tasks. For example, if a developer needs to meet non-functional requirements, they might care more about the trade-offs between the API methods. Future work should consider observing participants beyond our study settings, in real-world tasks with more diverse domains and APIs.

The observations gathered through interviews only represent the experience of the participants, most of whom were graduate students conducting ML/NLP research. The usefulness of information on comparable API methods and tools similar to ours may be different for other user groups, e.g., professional software engineers. However, we believe that many of our findings can be relevant to software engineers, as most (12/16) of the participants have experience working at large corporations as (ML) software engineers or research scientists.

V. Implications for Automation

Overall, our annotation effort (Section III) and user study (Section IV) showed that while developers generally find comparable API methods and the summaries useful in understanding the design space of APIs, such information is not easily available to the users and requires significant effort to manually collect. To support more use cases in TensorFlow (and ultimately, other libraries), we suggest using ML-based techniques to increase the efficiency compared to fully manual annotation. We expect that such techniques:

- *should maximize the utility of small hand-annotated datasets.* Extracting comparable API methods with explanations requires significant manual effort. Therefore, we recommend utilizing fine-tuning on top of models that are pre-trained (e.g., BERT [58]) on large, existing datasets (e.g., Wikipedia, Stack Overflow).
- *should prioritize extracting comparable API methods over summaries.* We observed that most of the participants used the provided comparable API methods to discover new API methods, or to navigate through the documentation, but only a few seemed to benefit from the summaries.
- *may prioritize improving recall over precision when extracting pairs of comparable API methods,* once it achieves a reasonable precision score. We observed that many of the participants were okay with having comparable API methods that might be less relevant to their tasks among the tooltip results, as long as the list contains at least some useful information.
- *may limit themselves to extractive over abstractive summarization.* Although it is likely more useful if one could *generate* a concise but informative summary (i.e., *abstractive* summarization), creating training samples for this purpose would substantially increase data annotation cost compared to an *extractive* summarization model, which only selects key pre-existing sentences. We note that excluding irrelevant sentences (for conciseness) is just as important as identifying relevant sentences, as Stack Overflow answers often cover multiple topics. Therefore, a model should evenly balance both precision and recall in extractive summarization.

VI. SOREL

In the previous section, we provided evidence for the benefit of knowledge extraction, and in particular presenting comparable API methods, to support developers' understanding of the API design space. We also derived requirements for an ML model to help automate this knowledge extraction process. In this section we present SOREL (Stack Overflow RELation extractor), an ML model that collects comparable API methods and relevant supporting natural language sentences from Stack Overflow answers.

To discover new facts, represented as triplets (entity 1, relation, entity 2) in unstructured text, one typically follows a two-pronged knowledge extraction approach. First, Named Entity Recognition (NER) is used to locate and classify entities. Relation Extraction (RE) then identifies relations of various kinds (including "no relation") between entity pairs.

In general, both problems are hard and the focus of active research in NLP, e.g., [74], [75]. In our work, since we are extracting only API method entities, we found that the NER step can be done accurately using pattern matching based on the full list of TensorFlow API symbols,[7] therefore we focused on the RE problem instead. In the RE step, we aim to identify the comparative relation "is comparable to" in an input Stack Overflow answer with more than one TensorFlow API call. To make the identified relations more useful, we also identify one or more sentences from the same answer, that offer the evidence supporting each identified relation. This provides a type of (extractive) summary for each relation. Therefore, our task is divided into two sub-problems:

- Comparative-relation extraction (RE) between entities.
- Supporting-evidence prediction (SEP).

A. Model Architecture

Figure 4 shows the architecture of our model, which is inspired by Yao et al.'s DocRED [52]. The model consists of four main components:[8]

BERT. We use a language representation model called BERT (Bidirectional Encoder Representations from Transformers) [58] to represent Stack Overflow answers. Due to its bidirectional nature, it provides deeper contextual information, which can help with document-level relation extraction. We used the pre-trained BERT model and a tokenizer from the original BERT paper provided via huggingface.[9][10] Due to the small amount of training data, we kept the BERT model weights frozen during the training, fine-tuning just the classification layers – a common approach when using small fine-tuning datasets [56], [57].

Relation Encoder generates representations for a pair of entities. Embeddings of the same entity which occur multiple

[7]In 50 randomly selected answers tagged with TensorFlow, the pattern matching approach missed just 12 out of 133 mentions of TensorFlow methods, achieving 91% recall and 100% precision.

[8]For ablations of the model design, see Table II.

[9]https://huggingface.co/bert-base-uncased

[10]We also considered BertOverflow [74], which was pre-trained with a Stack Overflow corpus, but found the original BERT to work better.

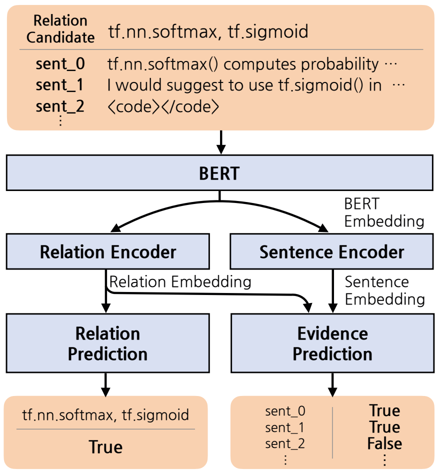


Fig. 4: The architecture of SOREL, which learns to infer the comparison relation and the supporting evidence.

times are averaged across the document, to allow sharing their learned representation across occurrences. The relation encoder combines two entities' averaged BERT embeddings using a bilinear function, which captures the mutual agreement between their respective representations. Similar to Yao *et al.*, we concatenated each entity representation with a relative distance embedding before passing it to the bilinear function [52]. The relative distance embedding represents the relative distances of the first mentions of each unique pair of entities in the document, informing the model of how closely together the two are mentioned. See [52] for details.

Sentence Encoder generates contextualized representations for each sentence in the input document. We use a two-layer bidirectional LSTM (BiLSTM) model to represent each token in the input document. BiLSTMs combine two LSTMs, one traversing the sequence forward and one backwards, to allow integrating information from the context on both sides of each token. This helps the model capture broader context around each sentence specifically for the sentence prediction task. An alternative would be to fine-tune the underlying BERT model, but we found this to be ineffective due to the small size of our training data (Table II). A single-layer, low-dimensional BiLSTM contains comparatively far fewer parameters to calibrate. To obtain a sentence embedding, we use the BiLSTM's output representation of the first token of sentence, which is the same (`[cls]`) token that BERT models use to extract sentence embeddings.

B. Model Training

As RE and SEP are highly connected tasks, we train SOREL RE and SEP objectives simultaneously, by combining the two losses: $loss = \alpha * re\_loss + sep\_loss$. We adjusted the hyperparameter α experimentally so the two losses converge at a similar rate, as minimizing the sep_loss involves more trainable parameters and thus typically takes more iterations.

TABLE II: Overall performance on each subtask, and ablations (on test set) of the model components and of the training set size (%).

	RE				SEP			
	A	P	R	F1	A	P	R	F1
Train	93.7	85.6	76.6	83.7	80.0	73.2	65.5	71.5
Val	89.3	64.3	89.6	67.9	76.2	65.9	61.6	64.7
Test	84.5	71.3	55.0	67.3	77.6	75.5	47.8	67.6
SOREL	**84.5**	**71.3**	55.0	**67.3**	**77.6**	**75.5**	**47.8**	**67.6**
- BERT	80.1	56.5	**59.3**	57.0	61.8	39.7	29.8	37.2
- BiLSTM	83.5	67.2	55.7	64.6	70.4	58.4	36.7	52.2
+ Finetune	80.9	58.6	58.6	58.6	69.8	57.3	34.0	50.4
All data (470)	84.5	71.3	55.0	67.3	77.6	75.5	47.8	67.6
2/3rd (315)	85.8	77.6	54.3	71.4	75.8	64.9	58.4	63.5
1/3rd (155)	84.5	73.0	52.1	67.6	73.9	81.1	27.6	58.5
1/10th (45)	78.8	53.3	64.3	55.2	65.9	45.9	17.9	35.0

TABLE III: Recall comparison with Google Autocomplete, Google's Top-5 results, TensorFlow documentation, heuristics, DiffTech [22], and APIComp [21]), on test set (%) after excluding deprecated API methods.

	SOREL	Auto	Top-5	Doc	Heur.	DiffTech	APIComp
RE	55.3	28.0	18.2	29.2	30.3	33.3	16.7
SEP	55.2	-	-	-	48.1	5.7	26.4

We randomly split our annotated dataset from Section III into a training and test set in a 4:1 ratio. We tuned the model hyper-parameters through 5-fold cross validation on the training portion (470 samples). In selecting hyper-parameters, when there are two settings with very similar overall performance, we selected the one that yielded a lower RE loss, under the premise that better comparable API methods extraction benefits a broader group of developers than having a good summary of the differences (as evidenced in Section IV-C).

To maximize the utility of our limited training data, we trained with a relatively low learning rate (1e-5) and frequently checked the held-out results to ensure that we captured the best performing model. We also added *input dropout*which randomly omits 40% of tokens from the input at training time [76]. This has the effect of preventing the model from overfitting on known inputs by artificially creating many versions of the same inputs, a form of data augmentation.

C. Evaluations with Test Data and Ablations

We now evaluate how well SOREL can extract knowledge on comparable API methods from Stack Overflow answers, focusing first on the test subset of our manually annotated data. Table II summarizes the train / (average) validation / test performance scores and ablations of the model components and the training set size. The final model achieves around 67% F1 score and around 80% accuracy on both tasks (RE and SEP) on the test set, which is a reasonable return given the small amount of training data.

Training custom word embedding instead of using BERT drops performance significantly on both RE and SEP. This confirms the benefit of adopting a large pre-trained language model when only a little training data is available. Not using a BiLSTM for SEP also reduced the performance significantly, which shows that SEP relies on this component to capture the global context beyond the initial representations offered by BERT. An alternative option, fine-tuning BERT along with our other parameters, decreased performance, as we might expect given the small size of the dataset.

Models show fairly steady gains in performance as we train on a progressively larger subset of the training data, starting from just 45 samples. While RE performance fairly quickly saturates, SEP performance progressively improves with more data. Still, we are content that a corpus of our size is right around the smallest size (and thus requires the least annotator overhead) where our model performance is adequately capable of generalizing, achieving balanced F1 results for RE and SEP and accuracy around 80% on each.

D. Comparison with Baselines and Prior Work

For the **RE** sub-task, we compare SOREL with: (1) general-purpose web search; (2) the TensorFlow reference documentation; (3) a simple heuristic approach; and (4) an adaptation of the most closely related prior work approach, DiffTech [22], which can identify pairs of similar technologies (e.g., *golang* and *javascript* for programming language) from Stack Overflow, a task which is conceptually similar to ours, albeit at a higher level of abstraction. See Table III for a summary of the results.

1) [RE] SOREL vs Google Search: Developers commonly find information on comparable API methods using a generic search engine. Given a pair of fully-qualified comparable API methods (A, B) in our test set, we enter the Google search query "A" and record whether B is mentioned anywhere within the top-5 Google search result landing pages; we then repeat the search in reverse, starting from "B". The recall measure we report for Google Search for the pair (A, B) is the average of the two. In addition, we separately inspect the Google search autocomplete suggestions for the query prefixes "A" and "A vs" (similarly "B" and "B vs") and check for the presence of B (A) among the suggestions. SOREL identified roughly twice as many relations as Google search in both cases.

2) [RE] SOREL vs Reference Documentation: Similarly, given a pair (A, B) we inspect the API reference documentation page for A and record whether it contains a link to B and vice versa, excluding all pairs involving a deprecated API call, which have no documentation page. Again, SOREL identified roughly twice as many relations as were mentioned in the official documentation.

3) [RE] SOREL vs Heuristic: Next, we compare SOREL to a simple, intuitive heuristic: if a Stack Overflow answer mentions exactly two API methods, consider those two to be comparable and record the pair. Across the 66 ground truth comparable pairs in test set, there are 52 answers with exactly two API methods mentioned, so one could hope to extract 52 comparable pairs. However, many of these pairs were not actually comparable (only 20 pairs were), so this simple heuristic is quite noisy, corresponding to 30.30% recall and 38.46% precision. Recall the example in Figure 1 – comparable pairs are often found within answers mentioning more than two API methods; the two API methods mentioned in answers with exactly two methods are often not comparable.

4) [RE] SOREL vs Word2Vec: We compare SOREL to DiffTech [22], the prior work closest to ours. DiffTech collects pairs of similar technologies (e.g., libraries, frameworks, programming languages) based on the intuition that frequently co-occurring Stack Overflow tags corresponding to different technologies may share a similar meaning. Specifically, DiffTech embeds tags with a Word2Vec [77] model trained on a corpus of tag sentences, and identifies pairs of tags as related when their embeddings have a cosine similarity greater than 0.4. While the DiffTech approach is designed to solve a different problem, the intuition carries over to our context: API methods may frequently co-occur with others they are comparable with, even though they rarely have dedicated tags. Therefore, we trained a Word2Vec model using API methods from our entire corpus of 33,460 TensorFlow answers and tested how many of the labeled comparable pairs in our test set have embeddings more than 0.4 cosine-similar. Only 16 (24%) of the comparable API method pairs in our labeled test set were among each other's top-5 nearest neighbors, and 22 (33%) were among the top-20, highlighting that our adaptation of the approach struggles to match comparable API calls based on co-occurrence statistics alone.

Next we turn to the **SEP** sub-task and compare SOREL to: (5) a heuristic approach; and (6) two approaches based on linguistic patterns, inspired by related prior work [21], [22].

5) [SEP] SOREL vs Heuristic: Intuitively, one might expect that given a pair of comparable API methods (A, B), all the sentences mentioning either A or B can be considered as supporting evidence. We test this heuristic for the true positive pairs from the RE step, for a fair comparison. In this limited setting, assuming the RE step is perfectly accurate, the heuristic approach performs quite well (Table III), but SOREL nevertheless manages to improve over it.

6) [SEP] SOREL vs Pattern Matching: Pattern-matching-based approaches have been used before to extract information from online documentation, with the two closest predecessors arguably being the DiffTech [22] approach discussed above for RE and APIComp [21]. A fundamental difference between SOREL and such approaches is where human effort is spent during the "training" process. SOREL requires human annotation effort to label API pairs and supporting sentences as valid / relevant or not. Pattern-matching-based approaches require human effort to identify the linguistic patterns needed to extract relevant sentences from text. Fundamentally, SOREL's type of effort can be expected to scale more easily to new APIs and sources of documentation, because it requires less expertise in NLP. Hence, we argue that designing a pattern-matching-based approach for our task *from scratch* may not be preferable. Still, one can ask a pragmatic question – what if the predefined linguistic patterns from DiffTech [22] and APIComp [21] generalize well enough to extracting supporting evidence for comparable API methods from Stack Overflow *without any additional effort*? We investigate this next.

For DiffTech [22], the authors identified and validated a series of linguistic patterns based on sequences of part-of-speech tags (e.g., "RBR (comparative adverb) JJ (adjective) IN (preposition)" as in "more efficient than") for extracting supporting sentences for the identified pairs of comparable technologies. Testing these same patterns on the answers in our sample containing 22 pairs identified by the previous RE step, we find that out of 70 sentences in 16 answers that were labeled as supporting evidence, only 4 sentences matched DiffTech's exact patterns (5%).

APIComp [21] is designed for a different usage scenario – to explain, given a pair of API methods, not necessarily "comparable" per our definition, the relationship between them using text extracted from reference documentation. APIComp first extracts sentences describing an API element from official reference documentation using linguistic patterns, and then aligns and compares the extracted sentences given an API knowledge graph. To test whether the same linguistic patterns carry over to our task, we applied the APIComp patterns to extract API statements from Stack Overflow answers in our test set, finding that only 33 out of 125 known supporting evidence sentences were matched (26.4%), identifying only 11 comparable API pairs out of 66 available (17%).

We conclude that previous pattern-matching-based approaches are not directly applicable to our task, and that anyway it may not be preferable to design a custom pattern-matching-based approach for relevant sentence extraction for our comparable API methods task. Recall is typically low because relations can be expressed in a variety of ways, hard to capture with reliable patterns, and pattern building typically requires significant amounts of manual effort.

E. Generalization to Larger Dataset

We next turn to our most ambitious target: assessing how well our model can generalize and rank insights from all 33,460 up-voted answers tagged with TensorFlow. After filtering out posts containing fewer than 2 Tensorflow symbols, 2,014 documents remained. Using our model, we identify 433 pairs of comparable API methods with 744 supporting evidence sentences. We report two such new (not included in our training data) relations in Table IV, as an example.

We manually reviewed the top 50 answers as sorted by the descending probability of both the relation and the supporting evidence sentences (according to the model's predictions, multiplied together). As many unique pairs of API methods were discovered in multiple Stack Overflow answers, we focused on relations detected twice or more, which we expect are more likely to be both accurate as well as relevant to developers. To compute the accuracy for RE on these, we follow the same guidelines as in Section III, marking relations correct when the relation between the two entities was factual and explicitly represented in the answer. Similarly, for SEP, we labeled samples correct when the predicted sentences explain the relation sufficiently without missing sentences. Note that

TABLE IV: A sample of new comparable API methods pairs and their supporting evidence extracted from Stack Overflow.

API Pair	**`tf.convert_to_tensor`**, **`tf.constant`**
Evidence	[SO Answer: 50981199] Each time a tensorflow operation expects a Tensor input but receives something else, it tries to convert it with `tf.convert_to_tensor`, and if successful, proceeds normally with that output. In case of a constant like 2, but also np.arrays, lists, tuples, or virtually any container or (well-formed) hierarchy of containers, `tf.convert_to_tensor` will create a Const operation, just like the one you would create by hand by calling `tf.constant`. `tf.constant` is nonetheless useful because it allows you, among other things, to name your constant in the graph and to control its data type (with name and dtype arguments respectively).
API Pair	**`tf.nn.embedding_lookup`**, **`tf.gather`**
Evidence	[SO Answer: 46440226] If `params` is a single tensor, the `tf.nn.embedding_lookup(params,ids)` operation treats `ids` as the indices of rows in `params`. If `params` is a list of tensors or a partitioned variable, then `ids` still correspond to rows in those tensors, but the `partition_strategy` (either `"div"` or `"mod"`) determines how the `ids` map to a particular row. Alternatively, you can use the `axis` argument to `tf.gather()` to select columns from `U`:

SEP is dependent on the results of the RE, the model cannot predict a correct supporting evidence sentence for a "faulty" relation or "no relation". Therefore, we only considered correct RE predictions when computing scores for SEP.

On these 50 relations extracted from top-scoring answers, the RE precision is 50% and SEP average accuracy is 77%.[11] Importantly, 90% of these correct relations were novel—not present in the training data. These results highlight that despite training on a small amount of data, a model like ours can show its strength when deployed on a very large corpus, where facts abound and high recall may be less important.

F. Error Analysis

Next we describes the patterns we observed in the mistakes the model makes. One common type of mistake was false positives due to lack of explicit topic change. Unlike API reference documentation pages, Stack Overflow posts often cover multiple topics in a short document. Therefore, the model sometimes failed to recognize the abrupt topic change and predicted a relation assuming that two entities were discussed in the same context. This error might be mitigated by preserving the original delineation of paragraphs and code blocks in the input data, though low textual consistency is a fundamental issue with Stack Overflow posts [6], [34].

Another common type of false positive is answers describing sequential use of API methods, such as ones explaining a code snippet line by line. As many of the labeled comparable API methods were described in sequence, the model often made wrong predictions in this case. This confusion is rather reasonable; when annotating the data, we occasionally found it challenging to determine whether methods in a pair were explicitly compared or simply mentioned in sequence.

[11]As there is no label for this data, we cannot compute other metrics for RE. For SEP, we averaged the accuracies of each answer.

Finally, code in text was a general issue for both RE and SEP. Although we filtered out code snippets, many answers contain code intermixed with their text. Our model may fail to capture such code's context and meaning, which results in some false-positive API method pairs predicted as having a relation whenever they are simply used together.

G. Threats to Validity

A threat to the external validity of our results is that we only used TensorFlow answers to build and evaluate our model. Therefore, our model might not generalize to other ML libraries such as PyTorch, or other domains such as Cloud or Graphics. While we do not expect other libraries and domains (with sufficient programmer interest) to have particularly less informative Stack Overflow answers, it remains a subject of future work to adapt our protocol to such use-cases.

We did not compare our model with state-of-the-art document-level relation extraction models built for general corpora such as Wikipedia [78], [79]. While we adopt a similar methodology, our corpus is both much smaller, less structured, and more domain-specific than the general Wikipedia corpus. We believe omitting such a comparison is reasonable as our goal was to explore and demonstrate the feasibility of exploiting Stack Overflow's relational insights using machine learning. As such, we do not claim that our model presents the best performance for this problem.

VII. Discussion

We showed that providing extracted comparable API methods can help developers understand the design space of APIs. We further built SOREL, which using relatively little labeled data can jointly extract relations and supporting evidence from new Stack Overflow answers. Next we present implications of our work for practitioners and researchers in this area.

a) ML-based Knowledge Extraction: To our knowledge, this is the first application of learning-based relation extraction, as well as supporting evidence prediction, on Stack Overflow data. The state-of-the-art knowledge extraction techniques we use make our pipeline a good fit for extracting structured knowledge from an environment as unstructured and diverse as Stack Overflow. A next step may be to annotate a corpus spanning multiple libraries, such as Pandas and PyTorch, following our protocol. Besides enabling SOREL for new libraries, this might also benefit performance overall: developers likely use similar patterns when comparing and contrasting terms from APIs, which could be useful to a model even when targeting just one API. Our results in Table II suggest prediction accuracy can yet improve with more data.

b) Reference Documentation: In the control condition (tooltips disabled) in our study, discoverability issues emerged due to a lack of "functional links" in the API documentation, that could only occasionally be retrieved via Google search, or had similar enough method names to be located nearby in the method navigation bar. However, when the reference documentation includes an explicit note about functionally-related methods, participants were able to discover them quite easily (see Section IV-C). Our detected comparable API methods could thus be especially useful for API documentation writers, to improve API discoverability. While automatically identified relations require the API to be popular enough on Stack Overflow and may include false positives or outdated pairs [8], [71], those well-versed in the API can identify correct relations in a list of extracted relations with little effort. This can quickly yield dozens of new functional links to boost API discoverability.

c) API Knowledge Support Tools: We believe it is worth developing other types of API knowledge acquisition tools like our prototype. One finding from our study that we believe might generalize is related to differences in problem solving and information foraging strategies between groups of users (we saw this especially around participant reactions to the summary sentences), which have also been noted previously [13], [80], [81]. This suggests that when designing knowledge support tools for developers, allowing users to configure the types and depth of information presented to them (e.g., whether to provide the summary) could be useful.

Another factor to consider is where the tool will be deployed. From the study, we found that the usefulness of our tool was highly dependent on Google search results, as most users activated the tooltip while on the Google search page. Some participants also suggested utilizing search queries so that they do not see methods that are less relevant to their tasks from the list (see Section IV-C for details). We believe that similar constraints (or benefits) will exist for other platforms (e.g., IDE), and thus, how users will consume the presented knowledge should be considered when designing the tool.

VIII. Conclusion

In this paper, we investigated the usefulness and challenges of extracting comparable API methods from Stack Overflow posts. From our user study, we provide evidence that showing comparable API methods and summaries about their difference can improve developers' understanding of the API design space, and help them select API methods by taking into account the differences between alternative solutions. We further showed that an ML-based model can reasonably accurately extract such knowledge from unstructured SO answers: our model identifies comparable API methods with a precision of 71% and summaries for these with 75% precision. We hope that our detailed observations and methodology inform further research on knowledge extraction in software engineering.

Data Availability. Our replication package, including our annotated dataset, annotation protocol, user study protocol, and scripts to replicate the user study, as well as our source code are available online at DOI 10.5281/zenodo.7570586 [82].

IX. Acknowledgement

This research was funded in part by the NSF under grant CCF-2007482. We would like to thank our participants for their time and input, and our reviewers for their valuable feedback.

References

[1] P. L. Li, A. J. Ko, and J. Zhu, "What makes a great software engineer?" in *International Conference on Software Engineering (ICSE)*, vol. 1. IEEE, 2015, pp. 700–710.

[2] K. Thayer, S. E. Chasins, and A. J. Ko, "A theory of robust api knowledge," *ACM Transactions on Computing Education (TOCE)*, vol. 21, no. 1, pp. 1–32, 2021.

[3] M. P. Robillard and R. DeLine, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[4] J. Stylos and B. A. Myers, "The implications of method placement on api learnability," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 105–112.

[5] E. Duala-Ekoko and M. P. Robillard, "Using structure-based recommendations to facilitate discoverability in apis," in *European Conference on Object-oriented Programming*. Springer, 2011, pp. 79–104.

[6] M. Liu, X. Peng, A. Marcus, S. Xing, C. Treude, and C. Zhao, "Api-related developer information needs in stack overflow," *IEEE Transactions on Software Engineering*, 2021.

[7] D. Nam, A. Horvath, A. Macvean, B. A. Myers, and B. Vasilescu, "MARBLE: mining for boilerplate code to identify API usability problems," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 615–627. [Online]. Available: https://doi.org/10.1109/ASE.2019.00063

[8] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online q&a forum reliable?: A study of api misuse on stack overflow," in *International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 886–896.

[9] X. Ren, J. Sun, Z. Xing, X. Xia, and J. Sun, "Demystify official api usage directives with crowdsourced api misuse scenarios, erroneous code examples, and patches," in *International Conference on Software Engineering (ICSE)*, 2020, pp. 925–936.

[10] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Statistical learning approach for mining API usage mappings for code migration," in *International Conference on Automated Software Engineering (ASE)*, I. Crnkovic, M. Chechik, and P. Grünbacher, Eds. ACM, 2014, pp. 457–468.

[11] M. Shaw, "Prospects for an engineering discipline of software," *IEEE Software*, vol. 7, no. 6, pp. 15–24, 1990.

[12] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: what do software developers want?" *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, 2018.

[13] A. Horvath, S. Grover, S. Dong, E. Zhou, F. Voichick, M. B. Kery, S. Shinju, D. Nam, M. Nagy, and B. Myers, "The long tail: Understanding the discoverability of api functionality," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2019, pp. 157–161.

[14] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE software*, vol. 20, no. 6, pp. 35–39, 2003.

[15] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow," Georgia Institute of Technology, Tech. Rep. GIT-CS-12-05, 2012.

[16] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 392–403.

[17] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography apis?" in *International Conference on Software Engineering (ICSE)*, 2016, pp. 935–946.

[18] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 643–652.

[19] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *International Conference on Program Comprehension (ICPC)*, 2014, pp. 83–94.

[20] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "Api method recommendation without worrying about the task-api knowledge gap," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 293–304.

[21] Y. Liu, M. Liu, X. Peng, C. Treude, Z. Xing, and X. Zhang, "Generating concept based api element comparison using a knowledge graph," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 834–845.

[22] H. Wang, C. Chen, Z. Xing, and J. Grundy, "Difftech: Differencing similar technologies from crowd-scale comparison discussions," *IEEE Transactions on Software Engineering*, 2021.

[23] C. Niklaus, M. Cetto, A. Freitas, and S. Handschuh, "A survey on open information extraction," in *International Conference on Computational Linguistics (COLING)*, 2018, pp. 3866–3878.

[24] 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.

[25] R. Abdalkareem, E. Shihab, and J. Rilling, "What do developers use the crowd for? a study using stack overflow," *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.

[26] 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.

[27] S. Nadi and C. Treude, "Essential sentences for navigating stack overflow answers," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 229–239.

[28] B. Kou, Y. Di, M. Chen, and T. Zhang, "Sosum: A dataset of stack overflow post summaries," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2022, pp. 247–251.

[29] X. Ren, Z. Xing, X. Xia, G. Li, and J. Sun, "Discovering, explaining and summarizing controversial discussions in community q&a sites," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 151–162.

[30] J. Sun, Z. Xing, R. Chu, H. Bai, J. Wang, and X. Peng, "Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 257–268.

[31] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger, "What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2258–2301, 2020.

[32] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 1295–1298.

[33] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stack overflow to turn the ide into a self-confident programming prompter," in *International Conference on Mining Software Repositories (MSR)*, 2014, pp. 102–111.

[34] G. Uddin, F. Khomh, and C. K. Roy, "Automatic api usage scenario documentation from technical q&a sites," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–45, 2021.

[35] S. Azad, P. C. Rigby, and L. Guerrouj, "Generating api call rules from version history and stack overflow posts," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, no. 4, pp. 1–22, 2017.

[36] J. Li, A. Sun, and Z. Xing, "To do or not to do: Distill crowdsourced negative caveats to augment api documentation," *Journal of the Association for Information Science and Technology*, vol. 69, no. 12, pp. 1460–1475, 2018.

[37] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving api caveats accessibility by mining api caveats knowledge graph," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 183–193.

[38] G. Uddin and F. Khomh, "Opiner: an opinion search and summarization engine for apis," in *International Conference on Automated Software Engineering (ASE)*, G. Rosu, M. D. Penta, and T. N. Nguyen, Eds. IEEE, 2017, pp. 978–983.

[39] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 548–559.

[40] P. Chatterjee, K. Damevski, and L. Pollock, "Automatic extraction of opinion-based q&a from online developer chats," in *International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1260–1272.

[41] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza, "Opinion mining for software development: A systematic literature review," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–41, 2022.

[42] R. Rubei, C. Di Sipio, P. T. Nguyen, J. Di Rocco, and D. Di Ruscio, "Postfinder: Mining stack overflow posts to support software developers," *Information and Software Technology*, vol. 127, p. 106367, 2020.

[43] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 706–716.

[44] R. F. Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixao, and M. de Almeida Maia, "Recommending comprehensive solutions for programming tasks by mining crowd knowledge," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 358–368.

[45] H. Wang, X. Xia, D. Lo, J. Grundy, and X. Wang, "Automatic solution summarization for crash bugs," in *International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1286–1297.

[46] A. Marques, N. C. Bradley, and G. C. Murphy, "Characterizing task-relevant information in natural language software artifacts," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 476–487.

[47] Y. Huang, C. Chen, Z. Xing, T. Lin, and Y. Liu, "Tell them apart: distilling technology differences from crowd-scale comparison discussions," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 214–224.

[48] S. Pawar, G. K. Palshikar, and P. Bhattacharyya, "Relation extraction: A survey," *arXiv preprint arXiv:1712.05191*, 2017.

[49] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.

[50] C. Wang, A. Kalyanpur, J. Fan, B. K. Boguraev, and D. Gondek, "Relation extraction and scoring in deepqa," *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 9–1, 2012.

[51] S. Kumar, "A survey of deep learning methods for relation extraction," *arXiv preprint arXiv:1705.03645*, 2017.

[52] Y. Yao, D. Ye, P. Li, X. Han, Y. Lin, Z. Liu, Z. Liu, L. Huang, J. Zhou, and M. Sun, "DocRED: A large-scale document-level relation extraction dataset," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019, pp. 764–777.

[53] K. Gábor, D. Buscaldi, A.-K. Schumann, B. QasemiZadeh, H. Zargayouna, and T. Charnois, "Semeval-2018 task 7: Semantic relation extraction and classification in scientific papers," in *Proceedings of The 12th International Workshop on Semantic Evaluation*, 2018, pp. 679–688.

[54] C. Alt, M. Hübner, and L. Hennig, "Fine-tuning pre-trained transformer language models to distantly supervised relation extraction," *arXiv preprint arXiv:1906.08646*, 2019.

[55] P. Shi and J. Lin, "Simple bert models for relation extraction and semantic role labeling," *arXiv preprint arXiv:1904.05255*, 2019.

[56] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[57] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799.

[58] 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.

[59] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining API mapping for language migration," in *International Conference on Software Engineering (ICSE)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 195–204.

[60] B. Collie, P. Ginsbach, J. Woodruff, A. Rajan, and M. F. O'Boyle, "M3: Semantic api migrations," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 90–102.

[61] A. Gokhale, V. Ganapathy, and Y. Padmanaban, "Inferring likely mappings between apis," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. IEEE Computer Society, 2013, pp. 82–91. [Online]. Available: https://doi.org/10.1109/ICSE.2013.6606554

[62] R. Pandita, R. Jetley, S. D. Sudarsan, T. Menzies, and L. A. Williams, "TMAP: discovering relevant API methods through text mining of API documentation," *J. Softw. Evol. Process.*, vol. 29, no. 12, 2017.

[63] N. D. Bui, Y. Yu, and L. Jiang, "Sar: Learning cross-language api mappings with little knowledge," in *Joint Meeting on the Foundations of Software Engineering (ESEC/FSE)*, 2019, pp. 796–806.

[64] A. Ni, D. Ramos, A. Z. Yang, I. Lynce, V. Manquinho, R. Martins, and C. Le Goues, "Soar: A synthesis approach for data science api refactoring," in *International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 112–124.

[65] "Automatically rewrite tf 1.x and compat.v1 api symbols," https://www.tensorflow.org/guide/migrate/upgrade, accessed: 2022-08-27.

[66] H. He, Y. Xu, Y. Ma, Y. Xu, G. Liang, and M. Zhou, "A multi-metric ranking approach for library migration recommendations," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 72–83.

[67] K. W. Nafi, M. Asaduzzaman, B. Roy, C. K. Roy, and K. A. Schneider, "Mining software information sites to recommend cross-language analogical libraries," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 913–924.

[68] C. Chen, Z. Xing, Y. Liu, and K. O. L. Xiong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 432–447, 2019.

[69] M. Carpentier, C. Combescure, L. Merlini, and T. V. Perneger, "Kappa statistic to measure agreement beyond chance in free-response assessments," *BMC medical research methodology*, vol. 17, no. 1, pp. 1–8, 2017.

[70] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[71] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 121–136.

[72] K. Hinkelmann and O. Kempthorne, *Design and analysis of experiments, volume 1: Introduction to experimental design*. John Wiley & Sons, 2007, vol. 1.

[73] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.

[74] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and named entity recognition in Stack Overflow," in *Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2020, pp. 4913–4926.

[75] Y. Huo, Y. Su, H. Zhang, and M. R. Lyu, "ARCLIN: automated API mention resolution for unformatted texts," in *International Conference on Software Engineering (ICSE)*. IEEE, 2022, pp. 138–149.

[76] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2627435.2670313

[77] 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 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 3111–3119. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[78] X. Han and L. Wang, "A novel document-level relation extraction method based on bert and entity information," *IEEE Access*, vol. 8, pp. 96 912–96 919, 2020.

[79] H. Tang, Y. Cao, Z. Zhang, J. Cao, F. Fang, S. Wang, and P. Yin, "Hin: Hierarchical inference network for document-level relation extraction," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2020, pp. 197–209.

[80] S. Clarke, "What is an end user software engineer?" in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[81] M. Meng, S. Steinhardt, and A. Schubert, "How developers use api documentation: an observation study," *Communication Design Quarterly Review*, vol. 7, no. 2, pp. 40–49, 2019.

[82] D. Nam, B. Myers, B. Vasilescu, and V. Hellendoorn, "Artifacts for Improving API Knowledge Discovery with ML: A Case Study of Comparable API Methods," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7570960

Appendix A
User Study Design

A. *Outcome Variables for Quantitative Analysis.*

Variable	Details
Task completion time (`time`)	We measured the time from when a participant first enters a search query to when they finish submitting the solution.
Number of search queries (`queries`).	We counted the number of search queries participants wrote until they submitted the final answer, as an approximation of how easy it was to retrieve necessary information.
Number of pages visited (`pages`).	We measured the number of web pages visited by a participant, to approximate the efforts needed in discovery.
Prior knowledge of the task (`prior`).	After each task, we asked participants whether they had implemented similar code in the past, and if so, whether that helped the search or not. We asked the level of prior knowledge in four levels: no experience (0), vaguely remembering that they have worked on similar tasks, but not enough to be helpful (1), experience with similar tasks helped them complete the task (2), and prior knowledge helped them recollect the exact function name (3).
The correctness of the solutions given the task definition (correctness).	We tested whether a participant submits an API method that matches with our ideal solution, which we determined based on our expert knowledge of the tasks. The tasks were purposefully designed to have multiple potential solutions, with varying implication and trade-offs, to test whether participants are aware of the alternative solutions. Thus, "incorrect" solutions may not necessarily be "bad" solutions, and they can still meet the task requirements with different quality implications, such as performance, readability, or scalability.
Awareness of the comparable API methods (`awareness`).	In each post-task interview, we tested whether a participant was aware of comparable API methods given the task context. When participants answered that they did not know other relevant API methods or admitted that they *guessed* the difference, we considered them to not be aware of the comparable methods. If they could elaborate on the differences, we considered that they were aware of the comparable API methods.
Understanding of difference (`understanding`).	In the post-task interviews, we also asked the participants to explain the differences between the comparable API methods. This is independent of the *awareness*, as one might already know about comparable API methods but not consider using that during the task. We measured the level of understanding with two levels: did not know the difference or guessed the difference (0), and describe at least one difference between the methods(1).

Appendix B
User Study Results

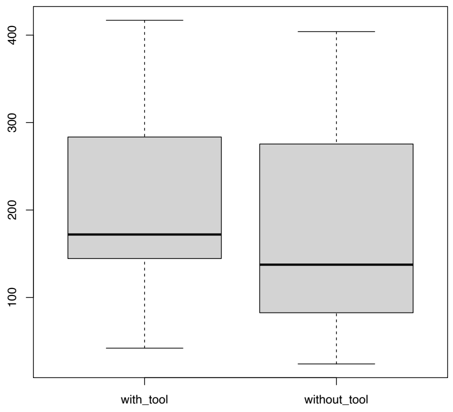
A. Quantitative Data Summary

Task ID	Our Ideal Solution	Comparable Methods	time		pages		queries		correctness	
			Tool	No	Tool	No	Tool	No	Tool	No
1	squeeze	reshape	153.50	69.75	2.00	1.00	1.25	1.00	0.75	1.00
2	einsum	[matmul, matmul]	171.75	223.00	3.00	2.75	1.75	2.25	0.00	0.00
3	concat	stack	139.25	123.75	1.25	2.25	1.25	1.00	1.00	1.00
4	boolean_mask	[where, gather]	282.00	120.75	2.50	2.00	1.25	2.00	1.00	0.75
5	floordiv	[divide, floor]	224.50	106.00	2.00	2.50	1.50	1.25	1.00	1.00
6	resize_with_pad	resize	222.75	158.25	2.50	1.75	1.00	1.25	1.00	0.75
7	conv1d	conv2d/convolution	239.25	298.75	3.50	2.50	1.50	1.50	0.50	0.25
8	sparse_softmax_cross-_entropy_with_logits	softmax_cross-_entropy_with_logits	217.00	255.50	2.00	2.25	1.00	1.00	0.75	0.50
Avg			206.25	169.47	2.34	2.13	1.31	1.41	0.75	0.66
SD			95.53	105.40	1.47	1.31	0.69	0.87	0.44	0.48

Task ID	Our Ideal Solution	Comparable Methods	awareness		understanding	
			Tool	No	Tool	No
1	squeeze	reshape	0.75	0.25	2.00	2.00
2	einsum	[matmul, matmul]	1.00	0.00	1.75	0.00
3	concat	stack	1.00	0.75	2.00	1.00
4	boolean_mask	[where, gather]	0.75	0.00	0.75	0.50
5	floordiv	[divide, floor]	1.00	0.50	1.75	1.50
6	resize_with_pad	resize	1.00	0.75	2.00	1.75
7	conv1d	conv2d/convolution	0.75	0.00	1.75	0.50
8	sparse_softmax_cross-_entropy_with_logits	softmax_cross-_entropy_with_logits	0.75	0.75	1.50	1.50
Avg			0.88	0.38	1.69	1.09
SD			0.34	0.49	0.64	0.96

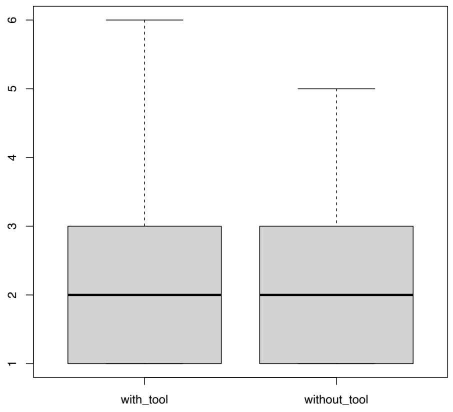
B. Statistical Test Results

1) Time:



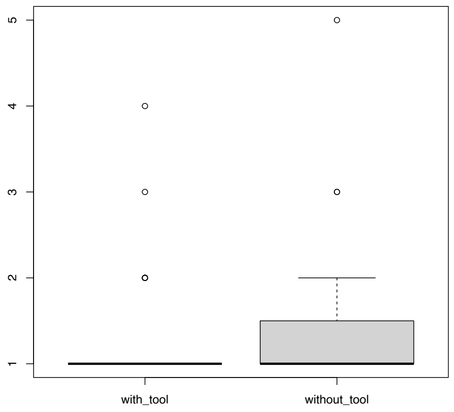
	Coeffs (Errors)
(Intercept)	198.69 (30.01)***
prior	-19.08 (14.35)
tool	37.97 (20.93)
AIC = 775.1; BIC = 788.1; LogLik = -381.5 $R2^m$ = 0.06. $R2^c$ = 0.30	

2) Pages:



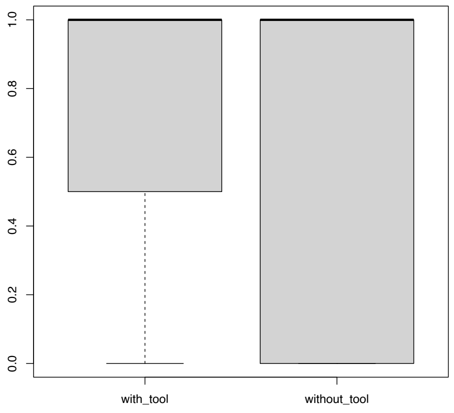
	Coeffs (Errors)
(Intercept)	2.36 (0.40)***
prior	-0.15 (0.20)
tool	0.23 (0.32)
AIC = 231.8; BIC = 244.8; LogLik = -109.9 $R2^m$ = 0.02. $R2^c$ = 0.14	

3) Queries:



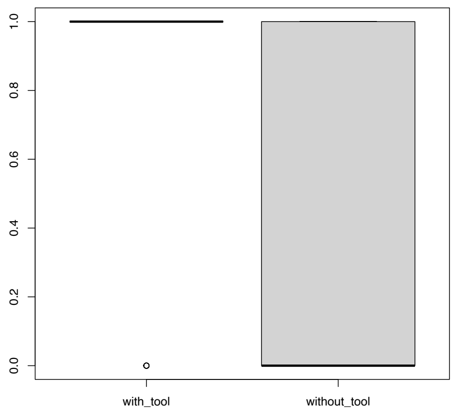
	Coeffs (Errors)
(Intercept)	1.41 (0.23)***
prior	-0.01 (0.12)
tool	-0.09 (0.18)
AIC = 160.1; BIC = 173.0; LogLik = -74.0 $R2^m$ = 0.00. $R2^c$ = 0.14	

4) Correctness:



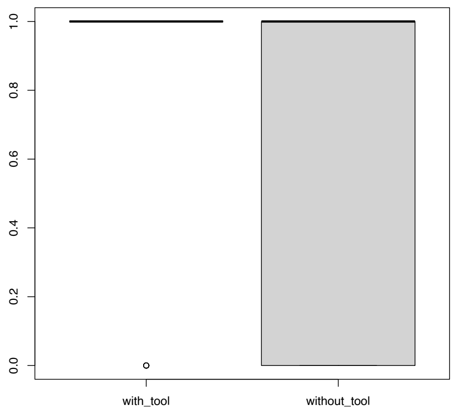
	Coeffs (Errors)
(Intercept)	-0.22 (1.50)
prior	0.96 (0.66)
tool	0.88 (0.87)
AIC = 66.2; BIC = 77.0; LogLik = -28.1 $R2^m$ = 0.07. $R2^c$ = 0.74	

5) Awareness:



	Coeffs (Errors)
(Intercept)	-1.12 (0.92)
prior	0.30 (0.44)
tool	3.03 (0.95)**
AIC = 73.7; BIC = 84.5; LogLik = -31.9 $R2^m$ = 0.35. $R2^c$ = 0.53	

6) Understanding:



	Coeffs (Errors)
(Intercept)	-1.08 (1.10)
prior	1.11 (0.58).
tool	2.63 (0.95)**
AIC = 64.8; BIC = 75.6; LogLik = -27.4 $R2^m$ = 0.31. $R2^c$ = 0.63	