Understanding Performance Concerns in the API Documentation of Data Science Libraries

Yida Tao
College of Computer Science and Software Engineering
Shenzhen University, China
yidatao@szu.edu.cn

Jiefang Jiang
College of Computer Science and Software Engineering
Shenzhen University, China
jiangjiefang2018@email.szu.edu.cn

Yepang Liu
Department of Computer Science and Engineering
Southern University of Science and Technology, China
liuyp1@sustech.edu.cn

Zhiwu Xu
College of Computer Science and Software Engineering
Shenzhen University, China
xuzhiwu@szu.edu.cn

Shengchao Qin*
School of Computing and Digital Technologies
Teesside University, UK
s.qin@tees.ac.uk

ABSTRACT

The development of efficient data science applications is often impeded by unbearably long execution time and rapid RAM exhaustion. Since API documentation is the primary information source for troubleshooting, we investigate how performance concerns are documented in popular data science libraries. Our quantitative results reveal the prevalence of data science APIs that are documented in performance-related context and the infrequent maintenance activities on such documentation. Our qualitative analyses further reveal that crowd documentation like Stack Overflow and GitHub are highly complementary to official documentation in terms of the API coverage, the knowledge distribution, as well as the specific information conveyed through performance-related content. Data science practitioners could benefit from our findings by learning a more targeted search strategy for resolving performance issues. Researchers can be more assured of the advantages of integrating both the official and the crowd documentation to achieve a holistic view on the performance concerns in data science development.

KEYWORDS

API documentation, performance, data science, empirical study

ACM Reference Format:
Yida Tao, Jiefang Jiang, Yepang Liu, Zhiwu Xu, and Shengchao Qin. 2020. Understanding Performance Concerns in the API Documentation of Data Science Libraries. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3324884.3416543

*Shengchao Qin is the corresponding author.

1 INTRODUCTION

Data science is an emerging field that combines mathematics, statistics, machine learning, and domain knowledge to derive insights from data [18]. As the volume and complexity of data grow rapidly, the painfully slow execution time and quickly exhausted RAM are becoming the major bottlenecks for developing efficient data science applications [30]. Since external data science libraries such as *Pandas*, *NumPy*, and *TensorFlow* are often used in these applications, their good performance is also vital for improving application efficiency and developer productivity.

Nevertheless, we observe persistent and active discussions on the performance problems of data science libraries. Take the Pandas library for example. As of April 2020, there are 853 questions on Stack Overflow that are tagged with both *pandas* and *performance*, and together these questions have over one million views. Furthermore, the answer acceptance rate for these questions is 58%, with an average of 41 days between the creation of questions and the proposal of accepted answers. On the GitHub repository of Pandas, 1,113 issues labeled with *performance* have been reported. While 83% of these issues are closed, the average resolution time is up to 134 days. These observations imply that data science developers may suffer from recurring interruptions caused by intractable performance problems. Existing work has also revealed performance issues in other data science libraries that cause runtime inefficiency [31, 42].

When data science developers are troubleshooting performance problems, documentation will likely be their first resort. For this reason, understanding how performance-related concerns are documented in data science libraries is crucial for steering effective performance optimizations. While previous studies have explored the quality [36], accessibility [23], and knowledge types [27] of API documentation, none of them targets on data science libraries and few focus on the performance aspects. To bridge this gap, we conduct an empirical study on popular data science libraries to understand the documentation practice on performance concerns, which denote descriptions and discussions related to runtime speed and memory consumption. In addition to the official API documentation maintained by library developers, we also consider two types of "crowd" documentation, Stack Overflow (SO for short) and

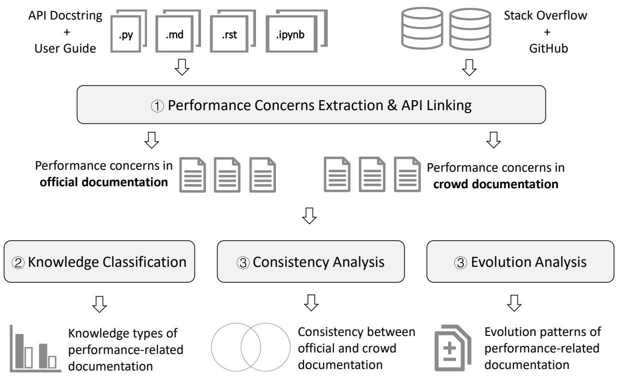

API Docstring + User Guide
.py
.md
.rst
.ipynb
Stack Overflow + GitHub
① Performance Concerns Extraction & API Linking
Performance concerns in official documentation
Performance concerns in crowd documentation
② Knowledge Classification
③ Consistency Analysis
③ Evolution Analysis
Knowledge types of performance-related documentation
Consistency between official and crowd documentation
Evolution patterns of performance-related documentation

Figure 1: Overview of our methodology.

GitHub, which are well-recognized resources for troubleshooting development problems [12, 14, 43]. The research questions we aim to address include:

- **RQ1**: How common are data science APIs documented in performance-related context? How does performance-related documentation evolve over time?
- **RQ2**: What types of knowledge are provided by performance-related documentation?
- **RQ3**: What are the differences between the official and crowd documentation in terms of performance-related content?

Our subjects include six widely used libraries from the Python data science ecosystem (Table 1). We leveraged program analysis, natural language processing routines, and traceability heuristics to automatically extract performance-related sentences from documentation and establish links between the sentences and corresponding APIs. We analyzed the extracted data to study the statistics of APIs that are documented in performance-related context. We also qualitatively analyzed the extracted performance concerns to understand their knowledge types, evolution reasons, and content consistency. Our major findings include:

- From 8.7% to 30.5% APIs of the target libraries have been documented in performance-related context, indicating the prevalence of performance concerns for data science libraries.
- Performance concerns in crowd documentation covers a different set of subject APIs compared to official documentation; only < 5% APIs have been mentioned in both data sources.
- Official documentation frequently provides the *function* knowledge (e.g., the runtime efficiency of a certain functionality), while crowd documentation focuses more on *practices* and *alternatives* types of knowledge. SO rarely provides explanatory knowledge types such as *purpose & rationale*.
- The vast majority (86.3%) of performance concerns from the crowd documentation provide new information that is not present in the official documentation; only a very few (2.7%) are inconsistent with the official documentation.
- Performance concerns in the official documentation are typically added long after the introduction of corresponding APIs, with an average of 0.48 follow-up updates that consist mostly of trivial semantic changes.

Table 1: Dataset statistics.

Library	Version	# APIs	# SO threads	# GitHub issues
NumPy	1.16.4	813	64,518	14,794
Pandas	0.24.2	1,828	118,636	29,387
SciPy	1.2.1	2,008	14,583	10,844
Scikit-learn	0.21.2	1,622	16,181	15,464
TensorFlow	1.13	3,635	46,142	33,738
Gensim	3.7.3	1,016	1,512	2,663

In general, we empirically analyzed how performance concerns on data science APIs are documented in terms of their occurrences, knowledge distributions, information consistencies, and evolution patterns. Our results further reveal how official documentation and crowd documentation complement each other in providing a holistic view of data science performance issues. We hope that our work, by providing a better understanding of the documentation practice on performance concerns, will benefit both data science practitioners and researchers in their battles against performance bottlenecks. Our data are released for public usage [41].

2 METHODOLOGY

In this section, we present our methodology in detail. The overall workflow is shown in Figure 1.

2.1 Data Collection

2.1.1 Target Libraries. As shown in Table 1, we focus on six widely-used libraries, all of which are core packages in the Python ecosystem for data science. These libraries solve different classes of problems and together support various important tasks in data science. In particular, NumPy and Pandas provide functionalities to work with arrays and tabular data [3, 4]; SciPy and Scikit-learn provide common numerical routines for statistics and machine learning [8, 9]; TensorFlow is a general-purpose deep learning framework [19] while Gensim is primarily used for natural language processing tasks [1].

2.1.2 Official Documentation. Official API documentation is typically authored and maintained by library developers. We consider two types of official documentation: the *API docstrings* and the official *user guide*. A Python docstring is a string literal, surrounded by triple double quotes, that appears as the first statement of an API definition. Developers often use docstrings to explain the general purpose of an API and how it should be used. Users can invoke Python's built-in `help()` function to check the docstring of a given API [6]. To collect the docstring data, we perform a depth-first traversal on the abstract syntax tree built from the root module of each target library to extract all the public APIs (i.e., *classes*, *functions* and *attributes*), as shown in Table 1. We then leverage Python's introspection capability [7] to obtain the file path, line range, and textual content of the docstrings for each API.

While docstrings provide succinct API-level descriptions, the official user guide provides a more high-level overview of a library. Users may find various types of information in a user guide, such as the basic concepts and algorithms used in the library, the tutorials and caveats on API usages, the guidance for extending the library, and so on. Compared to API docstrings, a user guide is more flexible in terms of the content, syntax, and structures. To collect the user guide data, we traverse the source directories of each library recursively to search for files with the format of reStructuredText (`.rst`), Markdown (`.md`) and Jupyter notebook (`.ipynb`). The former two file formats correspond to light-weight markup languages widely used for technical documentation, while Jupyter Notebook is an interactive computing interface often used for developing, debugging, and demonstrating data science applications [5].

2.1.3 Crowd Documentation. As online Q&A forums and social coding sites are reshaping the knowledge sharing in developer communities, research has proposed that sites like Stack Overflow and GitHub can be leveraged as essential complements to official API documentation [32, 39, 43, 44]. For this reason, crowd documentation is also considered as an important data source in our study. We used the official Stack Exchange REST API [11] and GitHub REST API [2] to crawl SO threads and GitHub issues of our target libraries. Note that an SO thread includes a question post and all of its answer posts and comments, while a GitHub issue includes an issue description and follow-up comments. Table 1 shows the details of our dataset.

2.2 Extraction of Performance Concerns and API Linking

Given the collected documentation, we identify performance concerns at the sentence level by first applying *spaCy* [10] for sentence segmentation. We then formulate a list of performance-related keywords based on literature [37] and common knowledge. The keywords include *fast, slow, expensive, cheap, performance, speedup, computation, accelerate, intensive, scalable, efficient* and their inflections (e.g., *efficiency, efficiently*).[1] Sentences containing these keywords are extracted as candidates that possibly discuss performance concerns.

Next, we identify the subject APIs that are being discussed in each candidate sentence, a problem often referred to as *API traceability* [16]. In this work, we used the *declaration-based*, *hyperlink-based*, and *mention-based* heuristics proposed in [23] to establish links between natural-language sentences and APIs. First, sentences from an API's docstring are directly linked to this particular API (i.e., *declaration-based*). Second, sentences from the official user guide and crowd documentation are linked to an API if they contain hyperlinks to that API's reference page (i.e., *hyperlink-based*). Third, we leverage regular expressions and the html tag <code> to extract explicit code mentions from candidate sentences in the official user guide and crowd documentation (i.e., *mention-based*). Note that a code mention may contain expressions or statements. To further identify the exact API(s) referred to in a code mention, we traverse its abstract syntax tree to search for `Call` and `Attribute` nodes. We then resolve each of these nodes to its fully qualified API names by matching it against APIs of the subject library, which is determined by SO tags and the identifier of the receiver object. For example, for a code mention `[row['a'] for _,row in df.iterrows()]`, we first extract `iterrows` as a function call. Since the code mention is from an SO thread tagged with *pandas* and the identifier `df` is a common abbreviation for `pandas.DataFrame`, we finally link it to the `pandas.DataFrame.iterrows` API. Sentences that cannot be linked or resolved to any of our target APIs are removed.

Note that sentences containing both the performance-related keywords and proper API links may not always address performance concerns. For example, "*the choice of ddof is unlikely to affect model* ***performance***" from the `sklearn.preprocessing.scale` docstring discusses the performance of machine learning models (e.g., accuracy) rather than runtime performance and memory consumption. The sentence "*convolve in1 and in2 using the* ***Fast*** *Fourier Transform method*" from `scipy.signal.fftconvolve` is also not related to performance although it includes the keyword "fast". To eliminate such false positives, we manually validated whether the automatically extracted candidates truly address API performance concerns (see Section 3.2 for details of the manual validation). Only true positive sentences were considered as valid performance-related documentation and used in subsequent analyses.

2.3 Knowledge Classification

After identifying performance-related documentation, we first analyzed the different types of knowledge provided by such documentation. To this end, we referred to the work of Maalej and Robillard, which proposed a taxonomy of 12 knowledge types derived from the API documentation of Java SDK 6 and .NET 4.0 [27]. While this taxonomy has been developed to be meaningful and reliable, it is unclear whether it can be directly applied to our task. A major concern is that the taxonomy was originally proposed to characterize general API documentation, whereas our goal is to characterize documentation specific to performance concerns.

To address this issue, we applied the *inductive coding* method [13] to better adapt the taxonomy to our performance-specific data. Initially, the first author used the original taxonomy to classify 100 randomly selected performance-related sentences. As part of this process, the first author removed existing knowledge types that are no longer applicable and added new knowledge types emerged from our data. The second author then repeated this process using the adjusted taxonomy. The third author helped reconcile the disagreement once the first two authors had different opinions of the taxonomy. This process continued until the new taxonomy was fixed.

Table 2 shows the details of our new taxonomy, which includes 11 knowledge types. Among which, six are directly reused from [27] and five are newly emerged from our data. In particular, we isolated the "Performance Attributes" knowledge type from the original "Quality Attributes and Internal Aspects" knowledge type. We merged "Internal Aspects" with the original "Control-Flow" and "Structure" types to form the new "Implementation or Internal Aspects" knowledge type. We introduced the "Usage Practice" knowledge type, which combines the original "Patterns" and "Code Examples" types since both describe how APIs should be used in practice. We also added the "Alternatives" knowledge type that is observed exclusively from our performance-specific data. Accordingly, each performance-related sentence was labeled with one or more knowledge types from this taxonomy.

[1] Inflection denotes different forms (e.g., noun, adjective, adverb) of a word.

Table 2: Taxonomy of knowledge types in performance-related documentation.

Knowledge Type	**Description**	**Example**
Performance **Attributes**	Describes the performance of an API, or the performance implications of different arguments or dataset.	Setting to False will improve the performance of this method. (`pandas.DataFrame.set_index`)
Functionality & Behavior*	Describes what the API does (or does not do) in terms of functionality or features.	The `Series.align` method is the fastest way to simultaneously align two objects. (`pandas.Series.align`)
Implementation or Internal Aspects	Explains an API's internal implementation that is only indirectly related to its observable behavior.	...internally this version uses a much faster implementation that never constructs the indices and uses simple slicing. (`numpy.fill_diagonal`)
Purpose & Rationale*	Explains the purpose of providing an element or the rationale of a certain design decision.	The `vectorize` function is provided primarily for convenience, not for performance. (`numpy.vectorize`)
Alternatives	Describes a more/less efficient alternative of an API, or compares the performance between alternative APIs.	Mini-batch sparse PCA `MiniBatchSparsePCA` is a variant of `SparsePCA` that is faster but less accurate (`sklearn.decomposition.MiniBatchSparsePCA`)
Usage **Practice**	Describes common usage scenarios for an API. Describes good, recommended or bad API usage practices.	To construct a matrix efficiently, make sure the items are pre-sorted by index, per row. (`scipy.sparse.lil_matrix`)
Concepts*	Explains the meaning of terms used to name or describe an API element, or describes design or domain concepts used or implemented by the API.	The main theoretical result behind the efficiency of random projection is the Johnson-Lindenstrauss lemma. (`sklearn.random_projection`)
Directives*	Specifies what users are allowed / not allowed to do with the API element. Directives are clear contracts.	We do require that your array be convertible to a NumPy array, even if this is relatively expensive. (`pandas.api.extensions.ExtensionArray`)
Environment*	Describes aspects related to the environment in which the API is used.	These modes will have different performance profiles on different hardware and for different applications. (`tensorflow.keras.layers.LSTMCell`)
References*	Includes any pointer to external documents, either in the form of hyperlinks, tagged "see also" reference, or mentions of other documents.	See the "enhancing performance" documentation for more details. (`pandas.eval`)
Miscellaneous	Includes other information such as TODOs and changes.	(TODO) Replacing or improving the performance of this would greatly speed things up. (`gensim.models.ldaseqmodel.sslm.update_obs`)

*Knowledge types proposed in [27].

2.4 Consistency Analysis

In addition to knowledge types, we also aim to understand the differences between official and crowd documentation in terms of the specific performance-related information they provide. To this end, we first matched API links to associate performance-related sentences from the crowd documentation with those from the official documentation that discuss the same subject APIs. Next, by comparing with the performance concerns in the corresponding official documentation, we manually classified each performance-related sentence from the crowd documentation as 1) *consistent*, if it provides the same or consistent information as the official documentation; 2) *inconsistent*, if it provides inconsistent or contradictory information as the official documentation; or 3) *not officially documented*, if the information it provides is not found in the official documentation.

It is worth noting that we originally planned to apply text similarity techniques to automate the abovementioned classifications. However, our preliminary experiment revealed that text similarity algorithms performed poorly on distinguishing information consistency, especially when the target sentences are already highly similar in terms of their subjects (i.e., they all discuss performance concerns on the same APIs). For example, two sentences with high similarity scores might convey completely contradictory ideas (e.g., "*This API is very fast*" and "*This API is very slow*"). Hence, we resorted to manual inspection for a more accurate analysis. Similar to that described in Section 2.3, the classification was conducted independently by the first two authors and discussed with the third author in case of discrepancies.

2.5 Evolution Analysis

Apart from the knowledge types and information consistency, characterizing how library developers evolve performance concerns is also important for understanding their documentation practice. In this study, we are interested in knowing when performance concerns are added to the documentation and how they are changed over time. For this purpose, we analyzed the commit history of performance-related sentences from the official documentation. Specifically, we characterized these commits in terms of 1) whether a performance concern is added to the documentation at the same time its subject API is introduced to the source code; 2) how often

Table 3: The number of performance-related sentences and corresponding APIs discovered from each data source.

Library	Docstring		User Guide		Stack Overflow		GitHub	
	# perf. sent.	# APIs	# perf. sent.	# APIs	# perf. sent.	# APIs	# perf. sent.	# APIs
NumPy	84	54	35	21	147	113	118	135
Pandas	65	53	115	79	164	96	150	121
SciPy	247	144	43	43	150	99	94	88
Scikit-learn	232	134	134	97	75	61	96	72
TensorFlow	161	120	67	50	122	102	97	93
Gensim	75	53	72	41	41	24	39	34
Total	864	558	466	331	699	495	594	543

performance-related concerns in the official documentation are modified; and 3) why a performance concern is modified.

We developed an algorithm that leverages the *git log* command, using the file path and line range as parameters, to extract all past commits of a given performance-related sentence. The algorithm uses similarity-based heuristics to automatically distinguish between the commit that *adds* the target sentence and the commit that *modifies* it. For a commit that adds a performance-related sentence, the algorithm checks whether it also contains the source file of the subject API and whether the API's definition is added to this file in this commit. If so, the commit is classified as *added together with APIs*, meaning that the API and its performance-related documentation are added and committed together; otherwise, the commit is classified as *added later*, meaning that the performance concern is added later after the API it refers to has been introduced in previous commits. In this case, we also analyzed the duration between the addition of the API and the addition of its performance concern. Note that the timestamp of an API/documentation addition was also extracted from the output of *git log*.

For a commit that modifies a performance-related sentence, we manually characterized the rationale of the change. We referred to a previous work that studied API documentation evolution [38] to initiate the taxonomy of evolution reasons. However, as this previous work focused on the general documentation of Java programs [38], its taxonomy could not be directly applied to our dataset. Therefore, we adjusted this taxonomy through an inductive coding process similar to that described in Section 2.3. As shown in Table 7, the adjusted taxonomy includes 11 reasons for updating performance concerns, six were rephrased from the findings of [38] and five were newly emerged from our data.

3 EVALUATION

In this section, we report the evaluation results on each step of our methodology.

3.1 Extracted Performance Concerns

Using the approach described in Section 2.2, we extracted 2,017 sentences that contain performance-related keywords from the official documentation. Among which, 1,330 were marked as true positives for truly addressing API performance concerns. As for the crowd documentation, we initially extracted 18,307 candidate sentences. To make the manual analysis feasible while statistically meaningful, we randomly selected 1000 sentences from SO and 1000 from GitHub for further processing. It is worth noting that the number of extracted candidate sentences for each library is different, with that of Gensim being the lowest. To balance the number of selected sentences across libraries, we set the probability of being randomly selected to be 0.2 for Gensim and 0.16 for the remaining five libraries. Among the 2000 candidate sentences from the crowd documentation, we identified 1,293 true positive sentences that indeed discuss performance concerns of our target APIs. Table 3 shows the number of performance-related sentences identified from each data source.

3.2 Manual Classifications

Our methodology requires manual efforts in four tasks, which include classifying the validity, the knowledge type, the information consistency and the evolution reasons of performance-related documentation. Manual work is crucial for identifying the complex nuances of natural language that often cannot be distinguished automatically. For example, sentences containing performance-related keywords may not really address performance concerns, and sentences with high similarity scores may not convey similar ideas (see examples from Section 2.3 and 2.4). Also, for text classification tasks, especially those newly-proposed and domain-specific ones (e.g., classifying the knowledge type of performance concerns, as proposed in this paper), manual labeling is often necessary before any automated method is attempted [20, 27].

Admittedly, manual classification is inherently subjective. To address this problem, each sentence was independently evaluated by the first and the second author as described in Section 2. Here we report the Cohen's kappa coefficient, which measures the agreement between two raters on a scale of 0–1 [28]. The kappa value for each of the four tasks is 0.89, 0.61, 0.71 and 0.63, respectively, meaning that the two raters had almost perfect agreement (kappa $\in$ [0.81,1]) for labeling the validity of performance-related sentences and substantial agreement (kappa $\in$ [0.61–0.8]) for labeling the knowledge type, information consistency and evolution reasons. Disagreements were reconciled with the third author joining the discussion.

3.3 API Linking and Commit Categorization

When we inspected each sentence to label its validity and knowledge types, we also checked the correctness and completeness of its API linking, which was automatically determined as described in Section 2.2. The precision and recall of our API linking approach

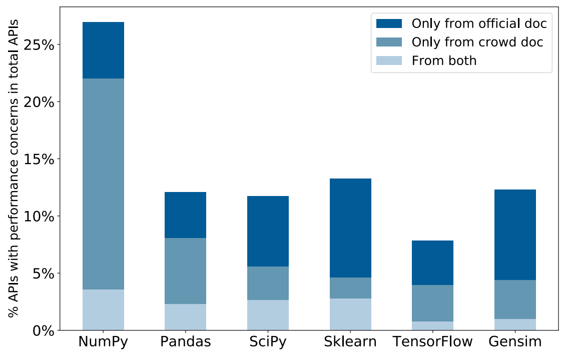


Figure 2: Percentage of APIs with performance concerns that are discovered from the official documentation, the crowd documentation, and from both data sources.

is 81.2% and 93.5%, respectively. False positives were mostly due to name ambiguity, when a name was incorrectly resolved to an API with different fully qualified name but having the same simple name. False negatives were mainly due to the absence of code-like characteristics. For example, our approach cannot extract the `sum` API when it appears as a common english word in a sentence without the `<code>` tag or API-indicating symbols like `()` in its surroundings. Since reliable API linkings are essential for studying API statistics (Section 4.1) and associating performance concerns related to the same APIs (Section 2.4), we manually fixed the incorrect and incomplete instances during this process.

We also evaluate our algorithm that automatically categorizes commit types in the evolution analysis (Section 2.5). We randomly selected 300 sentences and manually reviewed the results. Our algorithm achieves 96.3% accuracy in distinguishing commits of the *added together with APIs*, *added later*, and *modified* categories.

4 RESULTS

In this section, we report our study results with respect to each research question.

4.1 API Coverage

Table 3 shows the number of APIs discussed in the performance-related documentation of each library. Considering the total number of APIs shown in Table 1, all of the target libraries have nontrivial proportions of APIs being documented in performance-related context. Specifically, TensorFlow has 8.7% of its APIs being the subject of performance-related documentation while NumPy has up to 30.5% of such APIs. The other libraries have approximately 15% of their APIs being documented for performance-related reasons (Figure 2). This result manifests the prevalence of performance concerns for data science libraries.

By comparing the API names, we further analyzed the type of data source where each API is being discussed. As shown in Figure 2, each data source has its own merits. For NumPy and Pandas, the crowd documentation has performance-related discussions on more APIs compared to the official documentation. For the remaining libraries, however, performance concerns from the official documentation cover more APIs. Interestingly, only a few APIs are mentioned in both the official and the crowd documentation (< 5% as shown in Figure 2). These observations imply that SO and GitHub tend to complement official documentation by covering an additional set of APIs with performance concerns. We further expand on this issue in Section 4.3.

Finding 1: Performance concerns in crowd documentation cover a different set of APIs compared to official documentation. Together these two data sources have documented performance concerns for a nontrivial proportion of data science APIs.

4.2 Knowledge Types of Performance-related Documentation

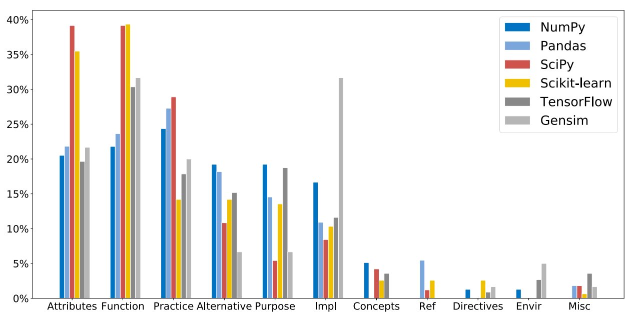
To better understand performance-related documentation, we analyzed the knowledge types of the extracted performance-related sentences using the taxonomy in Table 2. Figure 3 shows the knowledge-type distribution stratified by libraries and data sources. Below we highlight a few interesting observations.

Functionality is the most common knowledge type in API docstrings, yet it is not discussed often in crowd documentation. In particular, 33.4% of the performace-related sentences in API docstrings provide the *Function* knowledge, while the percentage of this knowledge type is 12.4% for SO and only 1.9% for GitHub. Since official documentation is inherently dedicated to describe functionalities and behaviors of APIs, library users, especially GitHub users who are often library developers themselves, may not find it necessary to include such duplicate knowledge again in their online discussions.

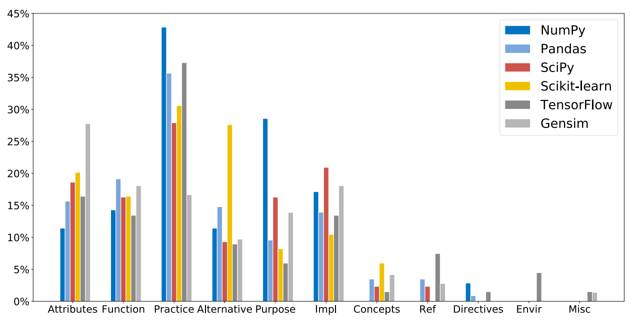
Usage practice is the top knowledge type discussed on Stack Overflow and official user guide. Specifically, the *Practice* knowledge type has a frequency of 36.6% on SO, 31.3% on official user guide, 21.7% on API docstrings, and only 9% on GitHub. These results suggest that for data science practitioners facing performance issues, SO and official user guide could be preferable to docstrings and GitHub for troubleshooting.

Crowd documentation provides more *Alternatives* type of knowledge compared to official documentation. In particular, around one fourth of the performance-related sentences from SO and GitHub have provided the *Alternatives* knowledge. A possible explanation is that unlike official documentation, which is inherently descriptive, crowd documentation is mostly in the form of information sharing and discussion. Therefore, users on the crowd platforms are more likely to discuss and compare the performance of alternatives in their problem-solving and bug-fixing activities.

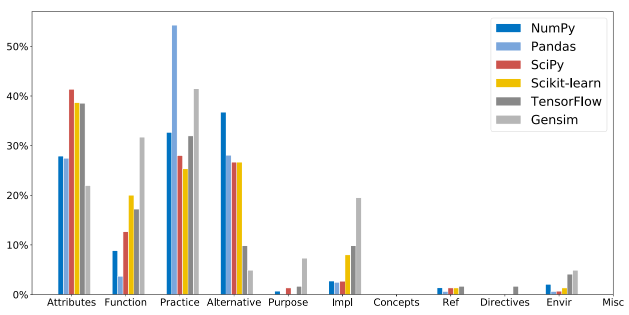
Stack Overflow rarely provides explanatory knowledge types such as *Implementation* and *Purpose*. Compared to official documentation and GitHub, which all have a certain degree of discussion on *Implementation* and *Purpose*, SO shows a surprisingly low frequency of these two explanatory knowledge types (5.4% and 1.1%, respectively). It is thus interesting to know whether SO focuses more on the "how" rather than the "why" in performance-related discussions, and if so, how such a tendency affects developers' learning and problem-solving skills. We consider this as a meaningful future research direction.



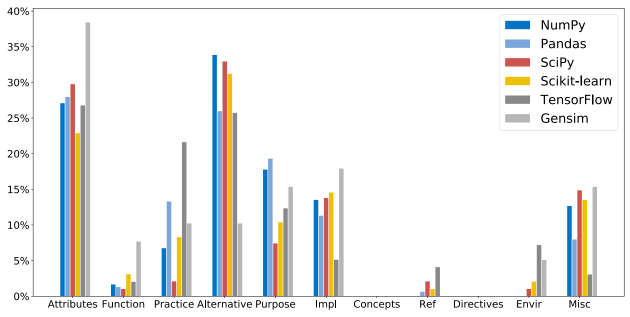
(a) API docstring



(b) User guide



(c) Stack Overflow



(d) GitHub

Figure 3: Knowledge types of performance-related sentences.

***{Function, Attributes}*, *{Practice, Alternatives}* and *{Impl, Purpose}* are knowledge types that often co-occur**. By analyzing the co-occurrence of knowledge types in performance-related sentences, we observed that developers tend to introduce API functionalities and performance attributes together, especially in the docstrings (e.g., *"Raising this value decreases the number of seeds found, which makes mean_shift computationally cheaper"* from Scikit-learn). Developers also tend to suggest preferable usage *practice* by comparing the efficiency of *alternatives* (e.g., *"You can either use fillna (fast) or you can use apply (slow but flexible)"* from SO post 42213549). Finally, *implementation* knowledge is often documented with explanations on the design *purpose and rationale* (e.g., *"This op expects unscaled logits, since it performs a softmax on logits internally for efficiency"* from TensorFlow). The confidence values for {Function}⇒{Attributes}, {Practice}⇒{Alternatives} and {Impl}⇒{Purpose} are 0.59, 0.35, and 0.27, respectively.

Finding 2: The *function* knowledge type is often observed in official documentation, while the *practice* and *alternatives* types of knowledge are more prevalent in crowd documentation. Explanatory knowledge types such as *purpose* and *implementation* are rarely observed on SO.

4.3 Information Consistency

In addition to API coverage and knowledge types, we also analyzed whether the specific information provided by crowd documentation is consistent with that in the official documentation (Section 2.4). Results are presented in Table 4 and detailed as follows.

Table 4: Consistency of performance-related information from crowd documentation w.r.t. official documentation.

	Consistent	Inconsistent	Not officially documented
SO	15.3%	3.4%	81.3%
GitHub	5.9%	1.8%	92.3%
All	11%	2.7%	86.3%

First, about 11% of the performance concerns in crowd documentation provide information that is consistent with the official documentation. In particular, SO has a larger percent of consistent information (15.3%) compared to GitHub (5.9%). Interestingly, these performance concerns, although being consistent, often have quite different narratives compared to the corresponding official documentation. Table 5 shows examples of such cases. Nonetheless, only ~5% of these consistent concerns have explicit references to the official documentation. It is thus interesting to know whether adding references to the official documentation could potentially shorten the resolution time of performance issues on crowd platforms.

Second, very few performance-related information from crowd documentation (2.7%) is inconsistent with the official documentation. Although this result seems promising, we observed that inconsistent instances typically convey contradictory information

Table 5: Examples of performance concerns from crowd documentation that are (in)consistent with the official documentation.

	Official Documentation	Crowd Documentation
Consistent	In fact, `pandas.eval` is many orders of magnitude slower for smaller expressions/objects than plain ol' Python. (`pandas.eval`)	`pd.eval('x // y', engine='python')` is 1000 times slower than the same operation in actual Python. (GitHub issue 4037)
	Please use `tf.keras.layers.CuDNNLSTM` for better performance on GPU. (`tf.keras.layers.LSTM`)	The CuDNNLSTM layer does the same as the LSTM layer in Keras, but it runs much faster on Nvidia GPUs (SO post 52340372)
	After the one-time initialization, a `Phraser` will be much smaller and somewhat faster than using the full `Phrases` model. (`gensim.models.phrases.Phraser`)	`Phraser` takes a bunch of extra time to create, but then is slightly faster but much more compact. (GitHub issue 837)
Inconsistent	`numpy.genfromtxt` is able to take missing data into account, when other faster and simpler functions like `numpy.loadtxt` cannot. (`numpy` user guide)	`genfromtxt` is a bit faster than `loadtxt` (SO post 52238949)
	Have a look at the `Hashing Vectorizer` as a memory efficient alternative to `CountVectorizer`. (`sklearn` user guide)	The great thing about `CountVectorizer` is that . . . , which makes it very memory efficient, and should be able to solve any memory problems you're having. (SO post 27339041)
	This implementation is by default not memory efficient because it constructs a full pairwise similarity matrix in the case where kd-trees or ball-trees cannot be used. (`sklearn.cluster.DBSCAN`)	I suggest you to try `sklearn.cluster.DBSCAN` - it has similar behaviour for some data (sklearn examples), also, it runs a way faster and consumes much less memory. (SO post 54533606)

with respect to time and memory consumption (see examples from Table 5). If we assume that official documentation is the ground truth, adopting inconsistent information from the crowd platforms could instead trigger unexpected performance degradation in data science applications.

Finally, the vast majority of performance concerns from the crowd documentation (86.3%) have not been found in the official documentation (Table 4). This indicates the potential value of crowd documentation for offering a large volume of new information on the performance of data science libraries. However, whether such unofficial information could really be leveraged as a reliable complement to official documentation depends on its quality (e.g., correctness and clarity). Therefore, classifying or characterizing the quality of unofficial performance information on crowd platforms could be a promising future research direction.

Finding 3: The vast majority of performance concerns from the crowd documentation are not present in the official documentation. Inconsistent performance-related information is rarely observed, while consistent information is often discussed on crowd platforms without referring to the corresponding official documentation.

4.4 Evolution of Performance-related Documentation

We now report the evolution patterns of performance concerns in the official documentation. First, for API docstrings, only 39.9% performance-related concerns are added to the documentation together with the corresponding API definitions. The majority of performance concerns (60.1%), however, are later added to the corresponding APIs' docstrings. Figure 4 shows that this trend is similar for all the target libraries except for TensorFlow. User guide data reveals a consistent yet more significant overall pattern: 92.1% of the performance concerns are added to the user guide after their

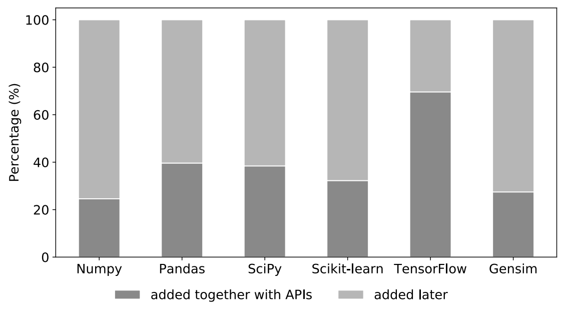


Figure 4: Percentage of performance concerns in the API docstrings that are added together with API definitions and added later.

corresponding APIs had been introduced to the source code. Considering all *added later* commits, we found that the average duration between the addition of an API and the addition of its performance concerns is 596 days. These results indicate that **developers tend to document performance concerns long after the addition of the subject APIs**.

Next, we study the update frequency of performance-related documentation. Table 6 shows the respective statistics for API docstrings and user guide. In general, the majority (73.1%) of performance concerns have stayed the same since they were added (i.e., zero updated times). Nearly one-fifth of the performance concerns (19.6%) have been updated just once. Only a few concerns (7.3%) have been updated multiple times (i.e., ≥2 updated times). On average, performance-related sentences have only 0.48 updates during library evolution. On the other hand, the subject APIs of these performance concerns have an average of 13.5 updates. These results indicate that **performance concerns are not updated often, whereas their subject APIs have been updated 28 times more frequently**.

Table 6: The number of times a performance concern in the official documentation is updated.

Updated times	# of performance sentences	
	API docstring	User guide
0	676	296
1	142	119
2	27	32
3	15	11
>3	4	8

Finally, we report the evolution reasons of performance concerns, which were observed from the 513 *modified* commits using the taxonomy in Table 7. As shown in Figure 5, most updates on performance concerns are formatting changes. Other common updating reasons include *clarification*, *improving fluency*, and *fixing spelling*. These results show that **developers typically apply trivial updates on performance-related documentation, without major changes to their semantics**.

In general, the evolution of performance-related documentation seems to lag behind the evolution of their subject APIs, in terms of both the creation time and maintenance activities. A natural question that arises here is whether performance-related documentation tends to be out-of-sync and potentially misleading for users to make performance-related judgement (see the example of wrong documentation in the "Fix wrong info" row of Table 7). We consider this as another research question that is worth further exploration.

Finding 4: Performance concerns are typically documented long after the addition of the corresponding APIs. Most of them have been stayed the same during library evolution or updated only once with mostly trivial semantic changes.

4.5 Implications

We now highlight a few important implications of our findings. For data science library users, our results suggest a more targeted strategy for searching performance-related information based on knowledge types (Finding 2). For library developers, keeping abreast of performance-related discussions on crowdsourced platforms might help them identify omissions in the official documentation (Findings 1–3). Also, more attention should be drawn on the consistency of performance-related documentation during the evolution of corresponding APIs (Finding 4). For researchers, investigating unofficial performance-related information on crowdsourced platforms (Finding 3) and rarely-updated performance concerns in official documentation (Finding 4) are both promising avenues for improving documentation quality. Research efforts could also be made on developing automatic techniques that merge performance concerns from official and crowd documentation while preserving the best of both worlds (Findings 1–3).

5 THREATS TO VALIDITY

Our study focuses on six Python libraries and two types of crowd documentation. The results may not generalize to other data science libraries, other programming languages, or other online forums and

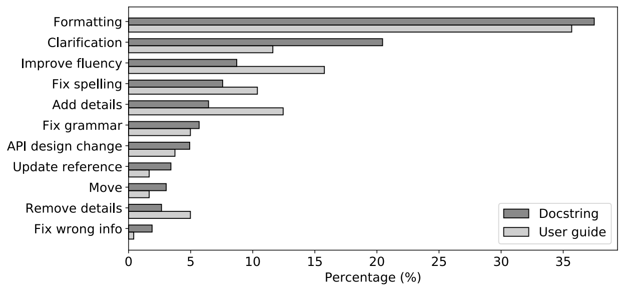


Figure 5: Reasons for updating performance concerns in the official documentation.

websites. To mitigate this threat, we took the diversity, representativeness, and popularity of the subjects into account in our subject selection process. The six libraries focus on different tasks of data science and are extremely popular among data science practitioners. Stack Overflow is the largest and most active online community for programmers, while GitHub is the mainstream platform for software development and version control.

We used performance-related keywords and code mentions to detect performance concerns for target APIs. However, this approach could not detect valid performance-related sentences that contain no predefined keywords and no explicit code mentions. Such false negatives may affect our quantitative observations. Nonetheless, the primary goal of this work is to study the documentation practice, in particular the knowledge types, information consistency, and evolution patterns, on performance concerns, rather than reporting all possible instances of such kind. The current methodology already detects sufficient amount of data for that purpose.

We proposed two taxonomies of performance-related documentation in Table 2 and Table 7. Admittedly, there could be other ways of categorizing the knowledge types and evolution reasons of performance concerns. To minimize this threat, we adapted related taxonomies of API documentation proposed in previous work [27, 38] rather than creating the taxonomies from scratch. Multiple human coders also iteratively adjusted the taxonomies. These processes helped improve the reliability and generality of our taxonomies.

Our study results might be affected by human subjectivity, which is a risk inherent in qualitative coding [13]. However, as described in Section 3.2, manual work is crucial for the type of tasks in this work. We followed the common research practice on manual labeling by inviting multiple experienced human raters and reporting the inter-rater agreement [23, 27]. Our results are also released for public scrutiny [41].

6 RELATED WORK

In this section, we discuss related work in the fields of API performance and API documentation.

API Performance: Performance of APIs and API usages has been studied in different domains. For Java applications, Kawrykow and Robillard observed cases where API call sequences can be replaced by more efficient ones [22]. Oliveira et al. proposed an

Table 7: Reasons for updating performance concerns in the official documentation.

Reason	**Description**	**Example (diff)**
API design change	Updating the documentation to reflect changes of API designs or API signatures (e.g., a new parameter is added, a function is renamed, a new algorithm is applied, etc.).	For small datasets, 'liblinear' is a good choice, whereas 'sag' is faster for large ones. For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. (*Scikit-learn commit 5147fd09*)
Add details*	Adding new information that is absent from the previous version.	This function is most efficient when 'n' is a power of two. This function is most efficient when 'n' is a power of two , and least efficient when 'n' is prime. (*SciPy commit e9250710*)
Clarification	Explicitly clarifying unclear concepts or providing supplementary notes, with only minor semantic changes and better understandability.	Provide this parameter to greatly speed up finite difference Jacobian estimation, if it's significantly sparse. Provide this parameter to greatly speed up finite difference Jacobian estimation, if it has only few non-zeros in *each* row. (*SciPy commit ad664040*)
Update reference*	Updating the reference or fixing broken links.	...will greatly speed up the computations [10]. ...will greatly speed up the computations [2]. (*SciPy commit 5ab0947e*)
Fix spelling*	Fixing spelling problems.	'lbgfs' can converge faster and perform better. 'lbfgs' can converge faster and perform better. (*Scikit-learn commit 79011904*)
Fix grammar*	Fixing grammatical errors.	...which is generally faster as "iterrows". ...which is generally faster than "iterrows". (*Pandas commit 89f04daa*)
Fix wrong info	Fixing incorrect or misleading information.	slower but more accurate alternative to NNDSVDa for dense NMF. faster, less accurate alternative to NNDSVDa for dense NMF. (*Scikit-learn commit eaced83a*)
Formatting*	Changing only the tabs, whitespaces, and markups to follow documentation conventions.	Allow overwriting data in 'a' (may enhance performance). Allow overwriting data in "a" (may enhance performance). (*SciPy commit ff48839a*)
Improve fluency*	Rephrasing the sentence to improve the presentation (e.g., word choices), typically with little or no semantic change.	Whether data in a is overwritten (may improve performance). Whether to overwrite data in a (may improve performance). (*SciPy commit c8ba75aa*)
Move	Moving the sentence to another location. The sentence itself is not changed.	(Diff is omitted for presentation concerns) (*TensorFlow commit 9dc48f95*)
Remove details	Removing notes and explanations that are unnecessary, obsolete, or expose too much details.	if 'True', use a faster, fused implementation based on nn.fused_batch_norm. if 'True', use a faster, fused implementation if possible. (*TensorFlow commit d2cf3938*)

*Reasons rephrased from [38].

approach that uses energy profiles and static analysis to recommend energy-efficient alternatives for Java collection implementations [29]. For Android apps, Linares-Vásquez et al. found energy bugs caused by suboptimal API choices [24]. Liu et al. identified performance issues caused by the misuses of the list scrolling API and wakelock APIs [25, 26]. Das et al. studied performance-related commits in Android apps and identified the most predominant types of performance-related changes [17]. For JavaScript programs, Selakovic and Pradel found inefficient API usage to be the most common root cause of performance issues [37]. For Rails applications, Yang et al. reported that half of the performance issues can be improved by changing how the Rails APIs are used [46]. Our work is different in that we study how API performance concerns are documented and we focus on the domain of data science.

Knowledge in Documentation: Researchers have studied the knowledge and information offered by software documentation from various perspectives. Pascarella and Bacchelli explored the code comments in Java projects and manually derived a hierarchical taxonomy of code comments in terms of their semantic meanings [33]. Hata et al. studied the content and evolution patterns of links in source code comments [21]. Prana et al. conducted a qualitative study to categorize the content of GitHub README files [35]. Li et al. mine API documentation using NLP techniques to build knowledge graphs of API caveats [23]. Maalej and Robillard proposed a taxonomy of knowledge types in the API documentation of Java SDK 6 and .NET 4.0 [27], which also inspires our taxonomy of knowledge types in performance concerns (see Section 2.3). However, previous studies tend to focus on the general content of API documentation, while our study focuses exclusively on performance-related content in the documentation.

Research has also suggested that crowdsourced knowledge can aid various software development activities. Regarding API understanding and usages, Petrosyan et al. used text classification to discover information that explains a given API but is scattered in online tutorials [34]. Treude and Robillard proposed a machine learning approach that classifies informative sentences from Stack Overflow to augment API documentation [43]. In this paper, we observed that official and crowd documentation complement each other in terms of performance-related information.

Documentation Quality: While software documentation greatly supports development activities, research has also found quality issues in documentation. Several studies combined program analysis and natural language processing techniques to detect inconsistencies between source code and documentation [40, 45, 48, 49], with the inconsistencies typically revealing incorrect or outdated information. Unlike these studies, our work compares performance concerns extracted from official documentation and crowd documentation. The reliability of crowdsourced data has also been studied. For example, Zhang et al. found that one third of SO posts contain potential API misuses [47]. Chen et al. found that a large proportion of security implementations on SO is insecure, and that the corresponding posts have higher scores and more duplicates compared to posts with secure suggestions [15]. In the future, we also plan to investigate the quality of unofficial performance-related information from crowdsourced data.

7 CONCLUSION

In this work, we present an empirical study on how performance concerns are documented in data science libraries. A nontrivial proportion of data science APIs was found to be discussed in performance-related context in the official and crowd documentation, indicating the prevalence of such problems. By quantitatively and qualitatively comparing the performance concerns extracted from these two types of data sources, we found that crowd documentation is highly complementary to official documentation in terms of the API coverage, the knowledge types, and the specific information being provided. We also observed that the maintenance on performance-related documentation is relatively plateauing and peripheral given the active evolution of their subject APIs. Our findings shed light on the current state of affairs for performance-related documentation practice, which could be an important step towards building efficient data science applications.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive suggestions and comments. This work was partially supported by the National Natural Science Foundation of China under Grants No. 61972260, 61772347, 61836005, 61932021, 61802164, and the Guangdong Basic and Applied Basic Research Foundation under Grant No. 2019A1515011577.

REFERENCES

[1] [n.d.]. Gensim. https://radimrehurek.com/gensim/

[2] [n.d.]. GitHub REST API v3. https://developer.github.com/v3/

[3] [n.d.]. NumPy. http://www.numpy.org/.

[4] [n.d.]. pandas. https://pandas.pydata.org/.

[5] [n.d.]. Project Jupyter. https://jupyter.org/

[6] [n.d.]. Python Docstring. https://www.python.org/dev/peps/pep-0257/

[7] [n.d.]. Python inspect - Inspect live objects. https://docs.python.org/3/library/inspect.html.

[8] [n.d.]. Scikit-learn. https://scikit-learn.org/stable/index.html.

[9] [n.d.]. The SciPy library. https://www.scipy.org/scipylib/index.html.

[10] [n.d.]. spaCy: Industrial-Strength Natural Language Processing. https://spacy.io/

[11] [n.d.]. Stack Exchange API v2.2. https://api.stackexchange.com/docs

[12] Eduardo C. Campos, Lucas B. L. Souza, and Marcelo de A. Maia. 2016. Searching Crowd Knowledge to Recommend Solutions for API Usage Tasks. *J. Softw. Evol. Process* 28, 10 (Oct. 2016), 863–892.

[13] Yanto Chandra and Liang Shang. 2019. *Qualitative research using R: A systematic approach.* Springer.

[14] Cong Chen and Kang Zhang. 2014. Who Asked What: Integrating Crowdsourced FAQs into API Documentation. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 456–459.

[15] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. 2019. How reliable is the crowdsourced knowledge of security implementation?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 536–547.

[16] Barthélémy Dagenais and Martin P Robillard. 2012. Recovering traceability links between an API and its learning resources. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 47–57.

[17] Teerath Das, Massimiliano Di Penta, and Ivano Malavolta. 2016. A quantitative and qualitative investigation of performance-related commits in android apps. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 443–447.

[18] Vasant Dhar. 2013. Data science and prediction. *Commun. ACM* 56, 12 (2013), 64–73.

[19] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[20] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 109–119.

[21] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 2019. 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay. In *Proceedings of the 41st International Conference on Software Engineering (ICSE'19)*. IEEE Press, 1211–1221.

[22] David Kawrykow and Martin P Robillard. 2009. Detecting inefficient API usage. In *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 183–186.

[23] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. 2018. Improving api caveats accessibility by mining api caveats knowledge graph. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 183–193.

[24] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining Energy-greedy API Usage Patterns in Android Apps: An Empirical Study. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, 2–11.

[25] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2014. Characterizing and Detecting Performance Bugs for Smartphone Applications. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. ACM, New York, NY, USA, 1013–1024. https://doi.org/10.1145/2568225.2568229

[26] Yepang Liu, Chang Xu, Shing-Chi Cheung, and Jian Lü. 2014. Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. *IEEE Transactions on Software Engineering* 40, 9 (2014), 911–940.

[27] Walid Maalej and Martin P Robillard. 2013. Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering* 39, 9 (2013), 1264–1282.

[28] Mary McHugh. 2012. Interrater reliability: The kappa statistic. *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB* 22 (10 2012), 276–82. https://doi.org/10.11613/BM.2012.031

[29] Wellington Oliveira, Renato Oliveira, Fernando Castor, Benito Fernandes, and Gustavo Pinto. 2019. Recommending Energy-efficient Java Collections. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, Piscataway, NJ, USA, 160–170.

[30] Carlos E Otero and Adrian Peter. 2014. Research directions for engineering big data analytics software. *IEEE Intelligent Systems* 30, 1 (2014), 13–19.

[31] Shoumik Palkar, James Thomas, Deepak Narayanan, Pratiksha Thaker, Rahul Palamuttam, Parimajan Negi, Anil Shanbhag, Malte Schwarzkopf, Holger Pirk,

Saman Amarasinghe, Samuel Madden, and Matei Zaharia. 2018. Evaluating End-to-End Optimization for Data Analytics Applications in Weld. *Proc. VLDB Endow.* 11, 9 (May 2018), 1002–1015.

[32] Chris Parnin and Christoph Treude. 2011. Measuring API Documentation on the Web. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering* (Waikiki, Honolulu, HI, USA) *(Web2SE'11)*. Association for Computing Machinery, New York, NY, USA, 25–30.

[33] Luca Pascarella and Alberto Bacchelli. 2017. Classifying code comments in Java open-source software systems. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 227–237.

[34] Gayane Petrosyan, Martin P Robillard, and Renato De Mori. 2015. Discovering information explaining API types using text classification. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 869–879.

[35] Gede Artha Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the Content of GitHub README Files. *Empirical Softw. Engg.* 24, 3 (June 2019), 1296–1327. https://doi.org/10.1007/s10664-018-9660-3

[36] Martin P. Robillard and Robert Deline. 2011. A Field Study of API Learning Obstacles. *Empirical Softw. Engg.* 16, 6 (Dec. 2011), 703–732.

[37] Marija Selakovic and Michael Pradel. 2016. Performance Issues and Optimizations in JavaScript: An Empirical Study. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. ACM, New York, NY, USA, 61–72. https://doi.org/10.1145/2884781.2884829

[38] Lin Shi, Hao Zhong, Tao Xie, and Mingshu Li. 2011. An empirical study on evolution of API documentation. In *International Conference on Fundamental Approaches To Software Engineering*. Springer, 416–431.

[39] Megan Squire. 2015. "Should We Move to Stack Overflow?" Measuring the Utility of Social Media for Developer Support. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 219–228.

[40] Lin Tan, Ding Yuan, Gopal Krishna, and Yuanyuan Zhou. 2007. /*icomment: Bugs or Bad Comments?*/. *SIGOPS Oper. Syst. Rev.* 41, 6 (Oct. 2007), 145–158. https://doi.org/10.1145/1323293.1294276

[41] Yida Tao, Jiefang Jiang, Yepang Liu, Zhiwu Xu, and Shengchao Qin. 2020. *Dataset for "Understanding Performance Concerns in the API Documentation of Data Science Libraries"*. https://doi.org/10.5281/zenodo.3972069

[42] Yida Tao, Shan Tang, Yepang Liu, Zhiwu Xu, and Shengchao Qin. 2019. How do api selections affect the runtime performance of data analytics tasks?. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 665–668.

[43] Christoph Treude and Martin P Robillard. 2016. Augmenting api documentation with insights from stack overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 392–403.

[44] Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. 2014. How Social Q&A Sites Are Changing Knowledge Sharing in Open Source Software Communities. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Baltimore, Maryland, USA) *(CSCW '14)*. ACM, New York, NY, USA, 342–354.

[45] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 53–64.

[46] Junwen Yang, Pranav Subramaniam, Shan Lu, Cong Yan, and Alvin Cheung. 2018. How Not to Structure Your Database-backed Web Applications: A Study of Performance Bugs in the Wild. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 800–810. https://doi.org/10.1145/3180155.3180194

[47] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are Code Examples on an Online Q&A Forum Reliable? A Study of API Misuse on Stack Overflow. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE'18)*. Association for Computing Machinery, New York, NY, USA, 886–896. https://doi.org/10.1145/3180155.3180260

[48] Hao Zhong and Zhendong Su. 2013. Detecting API documentation errors. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*. 803–816.

[49] Yu Zhou, Ruihang Gu, Taolue Chen, Zhiqiu Huang, Sebastiano Panichella, and Harald Gall. 2017. Analyzing APIs documentation and code to detect directive defects. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 27–37.