# Concept-Annotated Examples for Library Comparison

Litao Yan
Harvard University
Cambridge, MA, USA
litaoyan@g.harvard.edu

Miryung Kim
UC Los Angeles
Los Angeles, CA, USA
miryung@cs.ucla.edu

Björn Hartmann
UC Berkeley
Berkeley, CA, USA
bjoern@berkeley.edu

Tianyi Zhang
Purdue University
West Lafayette, IN, USA
tianyi@purdue.edu

Elena L. Glassman
Harvard University
Cambridge, MA, USA
glassman@seas.harvard.edu

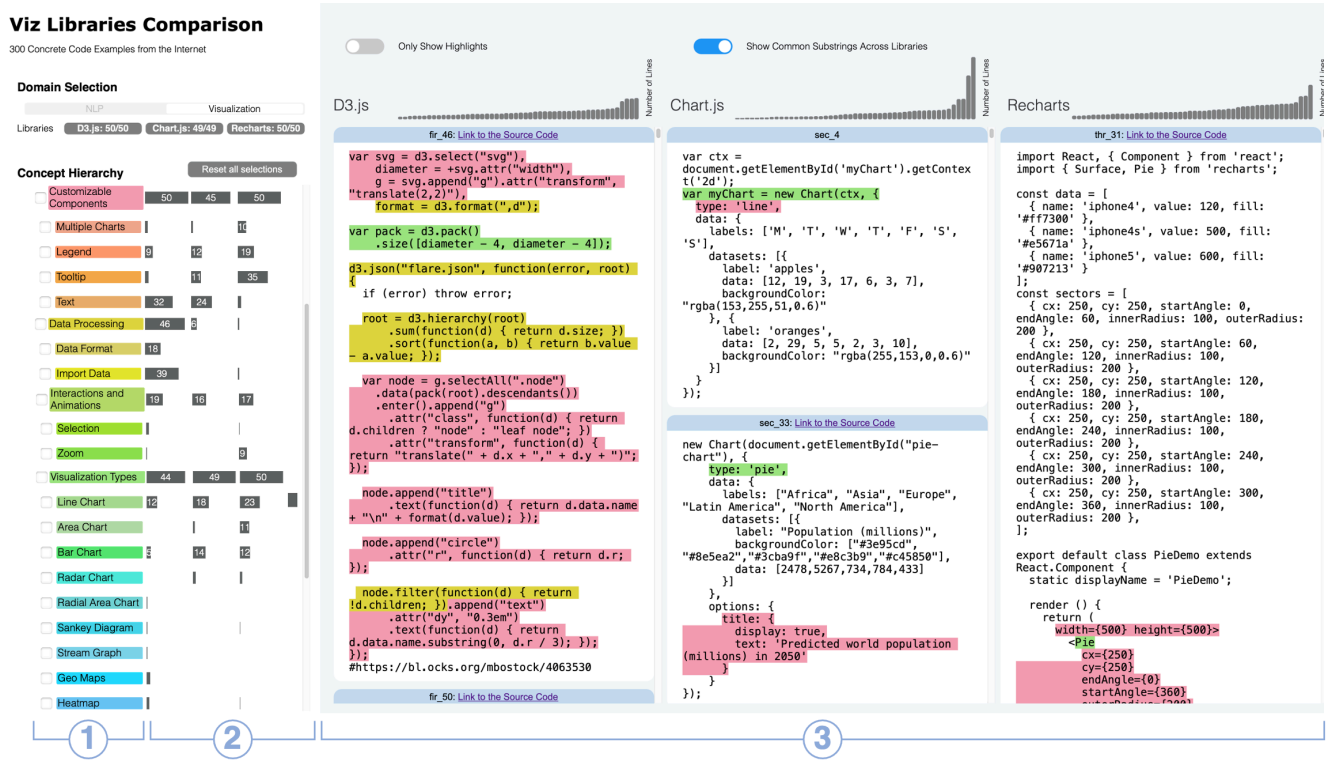


**Figure 1: ParaLib, an interactive interface for comparing and selecting suitable libraries with *snippets of code examples labeled according to a concept hierarchy.* ParaLib consists of three main components: (1) a hierarchy of concepts, e.g., interactions and animations : selection, over which to compare the functionality of each library, (2) the distribution of the number of examples from each library that contain code labeled with that concept, and (3) a side-by-side view of concept-labeled code examples from up to three different libraries.**

## ABSTRACT

Programmers often rely on online resources—such as code examples, documentation, blogs, and Q&A forums—to compare similar libraries and select the one most suitable for their own tasks and contexts. However, this comparison task is often done in an ad-hoc manner, which may result in suboptimal choices. Inspired by Analogical Learning and Variation Theory, we hypothesize that rendering many concept-annotated code examples from different libraries side-by-side can help programmers (1) develop a more comprehensive understanding of the libraries' similarities and distinctions and


*UIST '22, October 29-November 2, 2022, Bend, OR, USA*

ACM ISBN 978-1-4503-9320-1/22/10…$15.00
https://doi.org/10.1145/3526113.3545647

(2) make more robust, appropriate library selections. We designed a novel interactive interface, ParaLib, and used it as a technical probe to explore to what extent many side-by-side concepted-annotated examples can facilitate the library comparison and selection process. A within-subjects user study with 20 programmers shows that, when using ParaLib, participants made more consistent, suitable library selections and provided more comprehensive summaries of libraries' similarities and differences.

## CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**; **Interactive systems and tools**.

## KEYWORDS

Sensemaking; programming support; visualization

**ACM Reference Format:**
Litao Yan, Miryung Kim, Björn Hartmann, Tianyi Zhang, and Elena L. Glassman. 2022. Concept-Annotated Examples for Library Comparison. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3526113.3545647

## 1 INTRODUCTION

*"TensorFlow or PyTorch?"* Many beginner students of deep learning ask this question when they first start selecting a deep learning library to build their own neural network models. This type of selection task is common across many programming domains because of (1) programmers' pervasive use of third-party libraries, packages, frameworks, and APIs to reuse well-tested functionalities and (2) the increasing number and complexity of these third-party options. For example, there are now over 1.3 million packages on NPM, a JavaScript package manager, many of which have alternative packages with similar functionality [36]. Given many choices, it can be difficult for programmers to identify the library that is most appropriate for their own tasks and contexts [32].

The current de facto way programmers compare libraries is to search online and review documentation, blogs, and Q&A posts [9, 27, 50]. While there are several library selection websites, such as SaaSHub [43] and LibHunt [7], they do not provide affordances for *direct* comparison of libraries' supported functionalities, learnability, and usability. Liu et al. found that, when navigating online resources, programmers often go back and forth and compare multiple information sources, which is tedious [28]. They also found that this process is often non-linear and can become increasingly challenging as programmers continue to explore the decision space.

There are a number of specially built systems to support programmers' library comparisons and general software-related decision-making. Several published techniques have been proposed to support library comparison [10, 12], but they are limited to non-technical factors of libraries, e.g., popularity, release frequency, and issue response rate. Unakite [28] supports programmers in collecting and organizing information about decision-making trade-offs in software development, yet it requires users to manually build a comparison table with pieces of information from Stack Overflow.

In this work, we propose a novel interface prototype, ParaLib, that helps programmers compare and select libraries at the level of concrete code examples. We focus on four key dimensions identified in Vargas et al.'s need-finding study [27]: fit for purpose, size, complexity, and usability. These dimensions are revealed to users through the display of entire collections of *concept-annotated* code examples collected for each library being considered, rendered in *parallel* (Figure 1 ③), as well as empirical overviews, such as the distribution over concepts identified in the examples collected for each library (Figure 1 ②).

Key interface design choices were derived from the design implications of two theories of human concept learning: Variation Theory (VT) [34] and Analogical Learning Theory (ALT) [13]. VT suggests that many varying examples shown in parallel may help programmers form more robust conceptualizations of the libraries' similarities and differences in syntax and functionality, and require less working memory during a decision-making task than the sequential review of examples. ALT suggests that highlighting analogical concepts, i.e., functional correspondences, across all examples from all libraries will help the programmer appreciate—and see past—superficial differences.

Additional interactive features support sensemaking. For example, selecting concepts at any level of the functional concept hierarchy (Figure 1 ①) allows users to toggle the highlighting of individual concepts in each example and filter the collection down to a subset of examples that contain that concept. Collectively, these features support many of the tasks in Shneiderman's type by task taxonomy (TTT) of information visualizations [45], i.e., overview, filter, relate, and extract.

We designed and implemented ParaLib as a technical probe and evaluated it in a within-subjects user study (N=20) to measure the utility and usability of these theory-backed design choices. The results show that, compared to participants using online search, participants using ParaLib made more consistent, suitable library selections, provided more comprehensive summaries of libraries' similarities and differences, and reported more confidence in their choices. Since the example collections—nearly 50 examples per library—were hand-curated and annotated by the authors, the contribution of this paper is the interface design and the study of its effectiveness. Reaping the benefits of a ParaLib-like interface in a scalable way will require future investments in appropriate automatic or crowdsourced concept labeling, which we hope this paper motivates. In summary, our main contributions include:

- A theory-backed conceptualization of how large collections of concept-annotated code examples presented in parallel can support library comparison and assessment.
- The design and implementation of ParaLib, a novel interface that concretizes this conceptualization.
- A within-subjects user study that demonstrates the value of concept-annotated code examples, and motivates future work that assists in curating and annotating these collections.

## 2 BACKGROUND

Modern theories of human concept learning, like Variation Theory [34] and Analogical Learning Theory [13], describe the conditions under which humans form more accurate, robust conceptualizations. Variation Theory prescribes showing the human learner

sets of many examples, presented simultaneously, in parallel, that vary in certain ways, and are constant in others. This presentation of variation is designed to, and has been empirically validated to, help humans form more accurate conceptualizations of the object of learning that the examples do (or do not) represent. Analogical Learning Theory, meanwhile, prescribes identifying analogous structures or concepts across superficially different examples, ideally presented in parallel and aligned, so that the human viewer can more easily perceive the underlying shared characteristics, schemas, or mechanisms. The design of ParaLib is in part derived from these theories, in that it simultaneously shows multiple examples from each of the objects of learning (libraries under consideration) side-by-side and highlights analogical correspondences.

# 3 RELATED WORK

## 3.1 Empirical Studies on Library Selection

Selecting the right library or API is a critical step before writing code. Several studies have investigated factors to be considered when selecting libraries [1, 10, 20, 35, 37]. Vargas et al. [27] surveyed 115 developers and discovered 26 factors that influence the software libraries' selection. They found the size and complexity, fit for purpose, usability, and quality of match between a set of functionalities and the desired features are crucial to decision making. This motivates us to focus on four dimensions—fit for purpose, size, complexity, and usability, which are not yet supported by prior work for library selection.

Our focus on supporting library comparison at the level of concepts corresponding to key functionalities is motivated by previous studies on library and API learning [24, 33, 41, 42, 49]. Ko et al. [24] found that lacking code examples and the difficulty of determining the supported functionality are two of the six major learning barriers for programming interfaces. The API learning theory by Thayer et al. [49] also highlighted the importance of domain concepts in addition to code examples and usage patterns.

## 3.2 Tool Support for Library Comparison

Online library comparison websites such as StackShare [23] and LibHunt [7] provide high-level summaries of libraries' community support, e.g., number of GitHub stars and forks, crowdsourced pros and cons, the popularity of the libraries in helping developers complete various tasks, and online reviews written by library users. These online resources rarely include concrete code examples, or may only include a couple of examples that may not address the readers' informational needs.

The research community has also proposed several automated tools for library comparison [10, 12, 22]. LibComp is an IntelliJ plugin that supports library comparison based on quantifiable metrics such as popularity, release frequency, and performance [12]. Huang et al. [22] mine online discussions in Stack Overflow and clustered comparative sentences about similar software technologies. Instead of focusing on community support or quantifiable metrics, ParaLib provides a window into what the actual code looks like when using each library, as well as what functionalities of the library are used in practice.

Some prior work focuses on supporting users' annotation and evaluation of APIs and sometimes explicit comparison across multiple options. Adamite [21] provides many useful annotation features for API documentations. Unakite [28] allows users to add API-related information such as code snippets, user reviews, and tutorials to a table for comparison. Crystalline [30] automatically collects and organizes information in a tabular structure for decision-making when browsing the web. Strata [29] facilitates the reusing of prior users' programming knowledge. Unlike those tools, ParaLib displays many code examples, with features to support comparison, not the process of searching, collecting, and cleaning information.

While most of the existing comparison tools target API and library *users*, API and library *authors* may benefit as well: Zhang et al. [53] interviewed 23 API designers and found that they too want to compare APIs—specifically their own relative to alternatives made by others—to (1) identify which functionalities are supported by others but not theirs and (2) understand how well their own supports features provided by others.

There are several comparison techniques designed for other tasks and domains. Arab et al. developed HowToo for finding and sharing various programming strategies for the same problem [3]. Chang et al. [8] developed an interactive interface for customers to construct a comparison table to compare Amazon products based on user-chosen criteria and reviews. And additional tools have been created for comparing two workflows side by side [25], website design elements [26], and image manipulation tutorials [39].

## 3.3 Visualizing and Comparing Code Examples

Diff tools are a common utility in text editors and programming IDEs to render the differences between two or occasionally three code files, e.g., KDiff3 [11]. Distinct from these tools, ParaLib renders many tens of examples from each of multiple libraries and supports programmers' recognition of their similarities and differences.

Several techniques are designed to visualize many code examples at scale, but they are designed for tasks other than library comparison. OverCode [16] provides teachers with a high-level view of thousands of programming solutions through visualizing variations. Examplore [17] visualizes many code examples based on a pre-defined API usage skeleton to show the distribution over common and uncommon API usage choices in the wild for a single API call. ExampleNet [52] uses call graph analysis of a corpus of deep neural network source code to extract and visualize distributions over (1) neural network model architectures and (2) parameter settings.

# 4 INTERFACE DESIGN AND OVERVIEW

## 4.1 Design Goals

We set out five design goals for ParaLib, grounded in prior work on programmers' goals and information needs during library comparison:

- **D1. Code Examples:** Help users inspect a collection of concrete code examples when comparing multiple libraries [6, 44, 46, 52].
- **D2. Functionalities:** Help users compare the functionalities (and their associated syntax) provided by each library [14, 27, 32, 51].

- **D3. Usability:** Help users assess how easy it might be to use each library [2, 27, 35, 40].
- **D4. Size:** Help users anticipate the size of the code they need to write when using each library [27, 38, 51].
- **D5. Similarities, Differences, and Typicality:** Help users discover the functional and syntactic similarities and differences between multiple libraries, as well as some notion of respective typical usage from the distribution of code examples present in the collection [15–17, 52].

In other words, ParaLib is designed with the intention of making it easier for users to assess, relative to the other candidate libraries, what each library is capable of doing, what code leveraging each library typically looks like and how complex it is, and how easily they might be able to use it, as well as how customizable each library appears to be through the variation they see across all the examples available for each library.

## 4.2 Interface Overview

Based on these five design goals, we designed and implemented ParaLib, an interface backed by collections of hand-curated and annotated examples from multiple similar but distinct libraries; see Appendix A for details on how these example collections were produced. The following interface design choices were used to render these collections:

*Many Parallel Code Examples.* The code example view (Figure 1 ③) shows *many* distinct examples from each library under consideration, *side-by-side* with many examples from the other libraries. This choice is drawn from Variation Theory [34], as explained in Sections 1 and 2.

Users can scroll downward in any library column to reveal more examples. Each example is a complete code file, and above each example there is a link to its source, providing details on demand [45].

*Concept Annotations.* Using concept labels with uniquely assigned colors, ParaLib highlights conceptually analogous code segments across all examples from all libraries with the same color. For example, the snippets in all the examples annotated with Visualization Types are highlighted with the color of the block containing that concept label, i.e., green. The concept-to-color mapping is shown in the side panel (Figure 1 ①). This choice draws on Analogical Learning Theory [13] as elaborated on in Sections 1 and 2, and also addresses the *relate* task in Schneiderman's taxonomy [45].

Due to the large number of colors necessary for the many concepts in the hierarchies designed for this technical probe, users can hover over annotated code to disambiguate which concept the color highlighting refers to. When users hover over the name of each concept, a tooltip elaborates on what functionality it refers to.

Already, with these theory-derived design choices combined, users can use concrete code examples (**D1**) to directly compare code complexity and usability (**D3**, **D4**), functional and syntactic similarities and variations (**D2**, **D5**), and get a sense of typically utilized function calls (**D5**) across multiple libraries at the same time. Additional information visualization features further support users in their sensemaking:

*Concept Hierarchy.* The concepts listed in the side panel (Figure 1 ①) are organized into a hierarchy. Each row contains a concept identified by the data curator as either first-level (more general) or subordinate (more specific). The hierarchy groups related concepts. For example, in Figure 1 ①, the concept of Data Processing is a first-level concept that contains five second-level concepts: Data Format, Import Data, Data Transformation, etc. In order to initially provide users with a more general overview of the concept-annotated code, ParaLib defaults to only highlighting the code snippets corresponding to the more general first-level concepts in the hierarchy.

*Concept Distributions.* Three columns of bars (Figure 1 ②) form three distributions, one for each library under consideration, representing the number of code examples in each library's collection that include a snippet labeled with that concept. By comparing the length of these bars across the row for each concept, users can quickly see which functionalities are most commonly utilized in each library, if at all. The absence of a bar can indicate that the corresponding library may not support the concept (if the number of examples in the collection is large enough). With these concepts and corresponding distribution bars for each library, ParaLib provides an overview of possible functionalities and their typicality across multiple libraries.

*Example Length Distributions.* The distribution over the number of lines in each code example (top of Figure 1 ③) is rendered for each library. The same axes are used for these charts across all three libraries rendered in ParaLib so that users can directly compare the length of bars. When users traverse the code examples, the bars corresponding to the code examples within view are highlighted. Users can also click on any one of these bars to jump to the corresponding example that the user wants to look at, which might be particularly short or long.

*Concept Selection.* When a user selects one or more concepts by clicking their associated check boxes (Figure 2 ①), ParaLib filters out the code examples that do not include the selected concepts. For example, if a user wants to find examples that support both visualizing a line chart and adding an animation, she can click the check boxes for the concepts corresponding to line charts and animations.

As illustrated in Figure 2, in addition to filtering the collection of examples shown, other components of ParaLib also update when a user selects a concept. After a concept is selected, the distribution bars have two colors—the bars in dark gray represent the number of code examples after filtering, while the longer super-imposed light gray bars show the original number of examples in the collection prior to filtering (Figure 2 ④). If the user selects a second-level concept, the lines of code annotated with that second-level concept will change color from the first-level concept color to the subordinate second-level concept color (Figure 2 ③). The distribution over example lengths updates to reflect which examples are left (Figure 2 ②).

*Revealing Additional Within- and Across-Library Correspondences Between Examples.* Two additional buttons are intended to help users (1) focus on functionalities they most care about and (2) more easily see possible additional connections across libraries, beyond the concept annotations. The "Only Show Highlights" button,

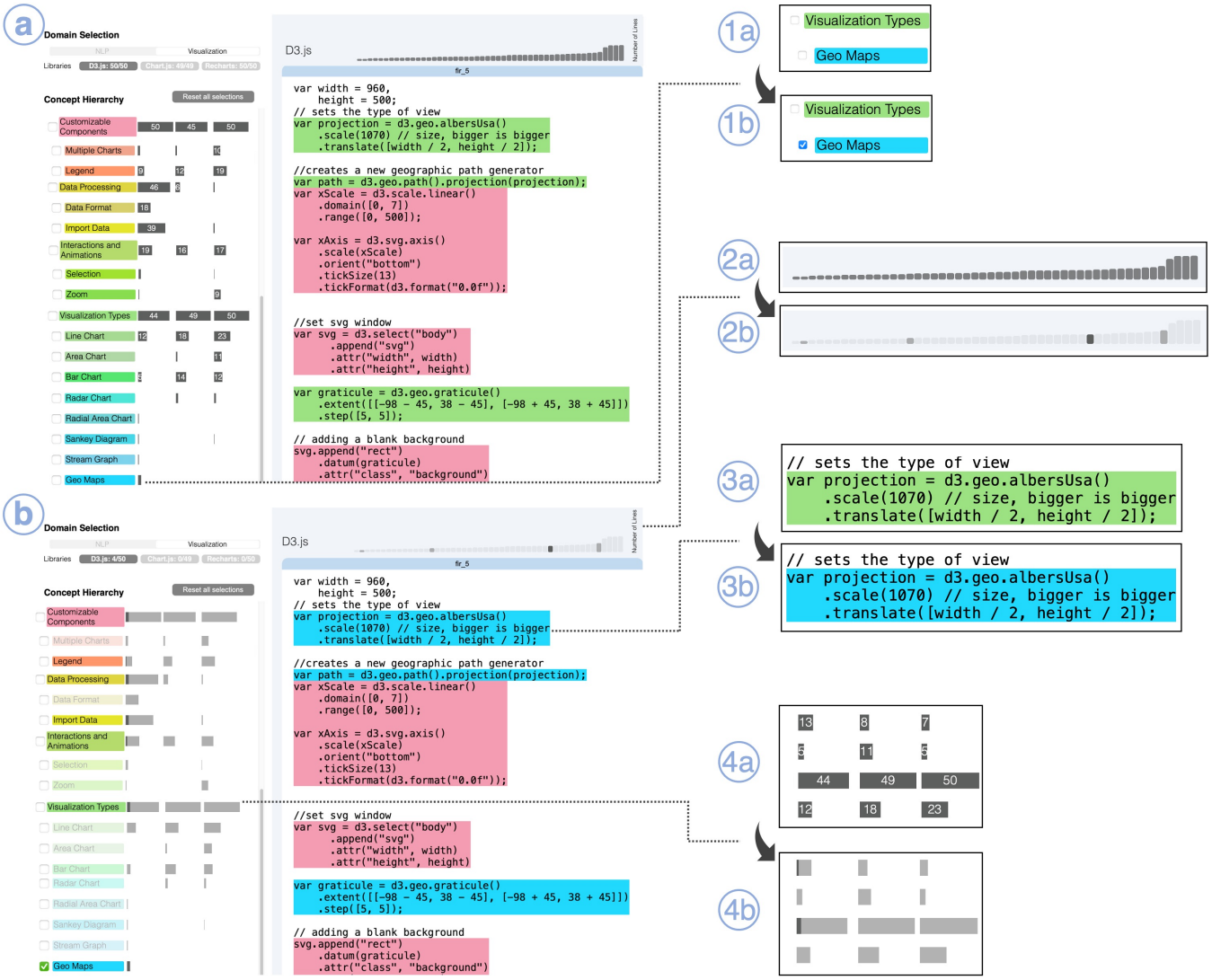


**Figure 2: After clicking on the Geo Maps concept (1a → 1b), ParaLib filtered out examples that do not contain Geo Maps functionalities. As a result, three components updated: example-length-representing bars corresponding to filtered examples were grayed out (2a → 2b); the color of code snippets annotated with the concept in the concept hierarchy changed from green, the color associated with their top-level concept annotation Visualization Types, to blue, the color associated with the more specific, selected second-level concept within Visualization Types: Geo Maps (3a → 3b); and the distributions over concepts represented in the remaining examples updated too (4a → 4b).**

shown most clearly in Figure 3 (1a-b), enables users to hide all unlabeled lines of code. With this view (Figure 3 (2a-b)), users hide code unrelated to the concepts they have chosen to focus on through default settings or explicit selections. The "Show Common Substrings Across Libraries" button, shown in Figure 3 (3a-b), increases the saliency of substrings that exist in multiple code examples from more than two libraries. Specifically, ParaLib bolds automatically identified similar function and variable names that are used by multiple libraries. For example, after checking the "lemmatization" concept, the substring `lemma` is shown in bold wherever it occurs in examples for all three libraries, shown in Figure 3 (4a-b).

## 4.3 Technical Probe Caveats

The hierarchy is currently designed by hand, in this case by the authors, and it is a function of the domain in which libraries are being compared. The method for constructing this hierarchy and the evaluation of its quality or how that quality impacts the user experience are beyond the scope of this work.

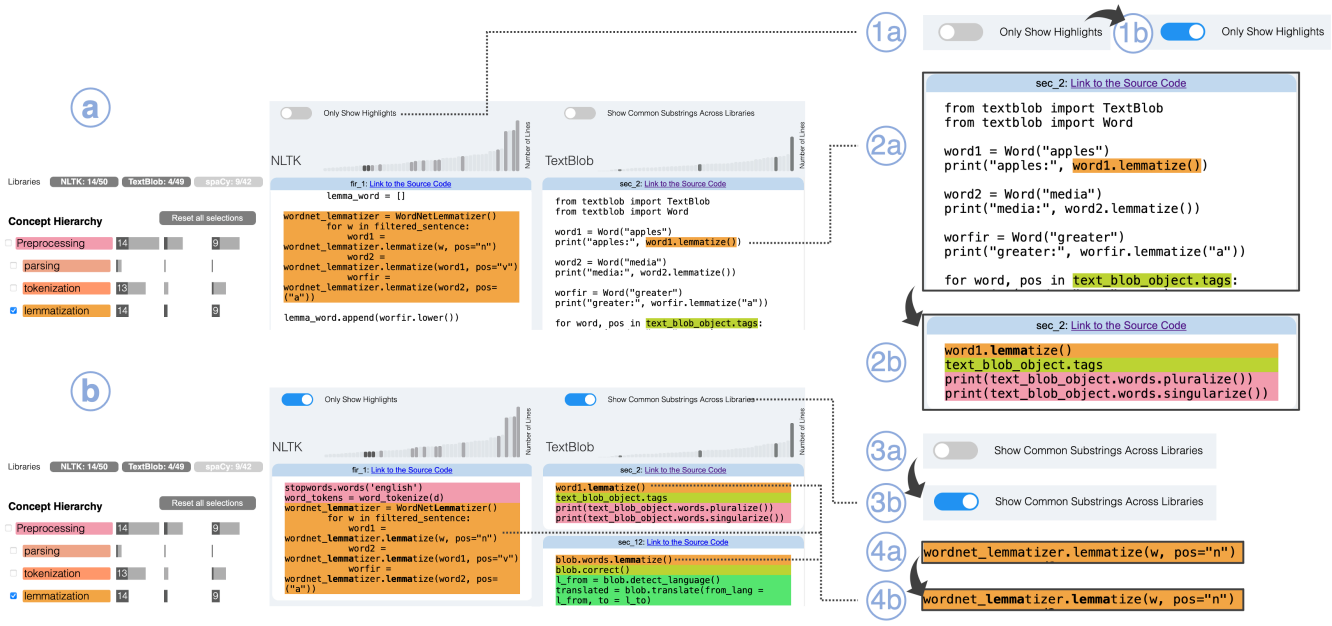


**Figure 3: Looking at multiple libraries' code examples related to the lemmatization concept: after checking `Only Show Highlights` button (1a → 1b), unrelated code was hidden (2a → 2b). After checking `Show Common Substrings Across Libraries` button (3a → 3b), the substring `lemma` was bolded in many examples across the two libraries shown (4a → 4b).**

Similarly, the information carried by the relative lengths of the distributions bars is assumed to scale with the quantity of the examples collected for each library, and the quality of the sampling method. For the purposes of evaluating the design choices implemented in ParaLib, we hand-created example collections of nearly 50 examples per library, as described in Appendix A.

## 5 USAGE SCENARIO

Jane, a software developer with access to ParaLib, is looking to visualize COVID-19 data in a treemap, but has not settled on a visualization library to use yet. She decides to consider `D3.js`, `Chart.js`, and `Recharts`, and ParaLib is loaded with concept-annotated example collections for each library.

Right away, Jane observes some interesting differences between libraries. Compared with `Chart.js`, which has a JSON-like format, `D3.js` looks more like a function-call-based library. `Recharts` has a syntax that is unlike anything else she has ever seen. By looking at the concept distributions, Jane can see at a glance which functionalities are more commonly used within each library. From these distribution bars, Jane also sees that, given the selection of examples in the corpus, `D3.js` supports the most functionalities, `Chart.js` only supports some common visualization types and functionalities, and `Recharts` is in between. Now, Jane gets a sense of the expressiveness of each library that's typically used in practice. She then sees that the distribution bar corresponding to the Treemapping concept in `Chart.js` is empty, which means that developers may not often use `Chart.js` to develop treemaps, or it may even be impossible because it is unsupported. Based on this, she decides to eliminate `Chart.js` from consideration for now.

After inspecting the lower-level concepts under the Visualization Types concept, Jane finds some visualization types she is unfamiliar with. By hovering over each of them, a tooltip shows a more detailed explanation of what it refers to. After reading these explanations, Jane is still most interested in making a treemap.

When Jane clicks on Treemapping, code examples that do not include Treemapping are filtered out of view, and the Treemapping-specific snippets of code change from the higher-level concept label's color, i.e., the color of Visualization Types, to match the color of the Treemapping concept, making it easier for her to visually pick out the snippets of code associated with the selected concept within the remaining examples.

Within this subset of examples, Jane wants to compare how she perceives each library's relative code complexity and potential usability. Looking across each library's example length distributions, Jane notices that the code examples for `D3.js` are typically longer than the other two libraries, though, at the long end of the distributions, `Chart.js` appears to catch up in length, for implementing non-trivial customizations. To reduce the visual noise of less relevant code, Jane checks the "Only Show Code Highlights" button, so ParaLib hides the unannotated lines of code, leaving just the concept-annotated code, which remains highlighted in concept-specific colors.

Since Jane finds the syntax of `Recharts` unfamiliar and difficult to parse, she eliminates it as well, selecting the more verbose but customizable `D3.js` to implement her treemap visualization of COVID-19 data. She scrolls down the collection of concept-annotated `D3.js` examples, seeing how there are recurring patterns in the sequence of concepts instantiated in each example, and the variety in which each concept is instantiated. This gives her a sense of the general structure that her code is likely to take, and what individual portions of it might look like, as she leaves ParaLib to write her own code.

## 6 USER STUDY

We conducted a within-subjects user study with 20 participants in order to evaluate ParaLib's effectiveness, i.e., how well it supports users' library comparison and selection processes. We hoped to answer the following questions:

- *What kinds of value does the concept hierarchy offer for comparing and selecting suitable libraries?*
- *How do concept-annotated code examples help programmers anticipate the complexity of code they are going to write with each library?*
- *In what ways do concept hierarchies and code examples help programmers develop useful insights?*
- *What is the user experience of using* ParaLib *to compare libraries like?*

### 6.1 Procedure

We designed two library comparison tasks: one for the domain of Visualization and the other for Natural Language Processing (NLP). In the Visualization domain, we selected three JavaScript libraries, `D3.js` [5], `Chart.js` [47], and `Recharts` [48], for participants to compare. In the NLP domain, we selected three Python libraries for participants to compare: `NLTK` [4], `TextBlob` [31], and `spaCy` [19].

Within each domain, the library comparison task included three components: First, ranking each library along five independent dimensions, i.e., (1) fit for purpose, (2) code size, (3) code complexity, (4) usability, and (5) appropriateness for a specific scenario. Second, listing the libraries' similarities and differences. Third, ranking their preference for using each library for a particular hypothetical scenario. See Appendix B for the exact wording of each component.

There were two conditions in which participants performed the library comparison task. In the control condition, participants completed the task using online search, i.e., any and all online resources, such as tutorials, blogs, Stack Overflow Q&A, and each library's official documentation. In the experimental condition, they were only allowed to use ParaLib to compare libraries. Prior to attempting to compare libraries in the experimental condition, every participant watched a five-minute tutorial video about ParaLib, and then had five minutes to interact with and familiarize themselves with ParaLib.

The entire study took about an hour. Each participant was given 20 minutes to compare each set of domain-specific libraries. As a within-subjects lab study, each participant compared both sets of libraries over the course of their lab session, completing the comparison task in one domain just using ParaLib and in another domain just using online search. Both the order of domains and conditions were counterbalanced.

After comparing each set of libraries, participants completed a questionnaire to record their reflections on their experience in the assigned condition. As part of the post-task survey, participants were asked to answer five NASA Task Load Index questions [18] to rate the cognitive load of the assigned task. After finishing both tasks, participants were asked to fill out another survey to directly compare the online search and ParaLib conditions.

### 6.2 Scenario Design

Based on real-world use cases, we created a library selection scenario for each domain. We designed each scenario so that only one library could meet all requirements. In the NLP domain, we constructed a scenario about using a pre-trained neural network model to classify tweets. The two requirements, i.e., the library should provide a pre-trained neural network and the library should be able to help with text classification, are only supported by `spaCy`. `NLTK` and `TextBlob` both include some important machine learning techniques, but neither provide a pre-trained neural network model. For the second requirement, all three libraries support text classification.

In the visualization domain, we describe a scenario for visualizing COVID-19 data on a map with two requirements: (1) the library has the ability to filter data and (2) the library supports geo-maps without the use of additional plugins. Given these constraints, `D3.js` is the unambiguous best choice because `Chart.js` does not support geo-maps and `Recharts` requires external plugins to produce geo-maps.

### 6.3 Participants

We recruited 20 participants (11 male, 9 female) from three participant pools: Eight participants were recruited from Reddit's *r/learnpython* and *r/learnjavascript* forums. Nine participants were recruited via the CS graduate mailing list at Harvard University, and 3 were recruited via the CS graduate mailing list at Purdue University. All of the participants were over 18 years old and fluent in English.

Participants had a range of prior experience in both Python ($\mu = 3.95, \sigma = 1.00$, on a 6-point Likert scale) and JavaScript ($\mu = 1.90, \sigma = 1.71$, on a 6-point Likert scale). Participants ranged from having one year to over five years of experience programming.

In order to capture the influence of participants' previous experience in each domain on the task results, we asked participants about their proficiency in both the visualization and NLP domains. In the data visualization domain, 40% indicated they had less than one year of experience, 50% had two-to-five years of experience, and 5% had over five years of experience. In the NLP domain, the majority of participants (11 out of 20) had learned NLP in less than one year, five participants had two-to-five years of experience, and one had more than five years of experience in the NLP domain.

Participants' prior library knowledge may influence their final library selection. Therefore, we asked about their familiarity with each library (`D3.js`, `Chart.js`, and `Recharts` in the visualization domain, and `NLTK`, `TextBlob`, and `spaCy` in NLP) on a 6-point Likert scale. They have various backgrounds in using the

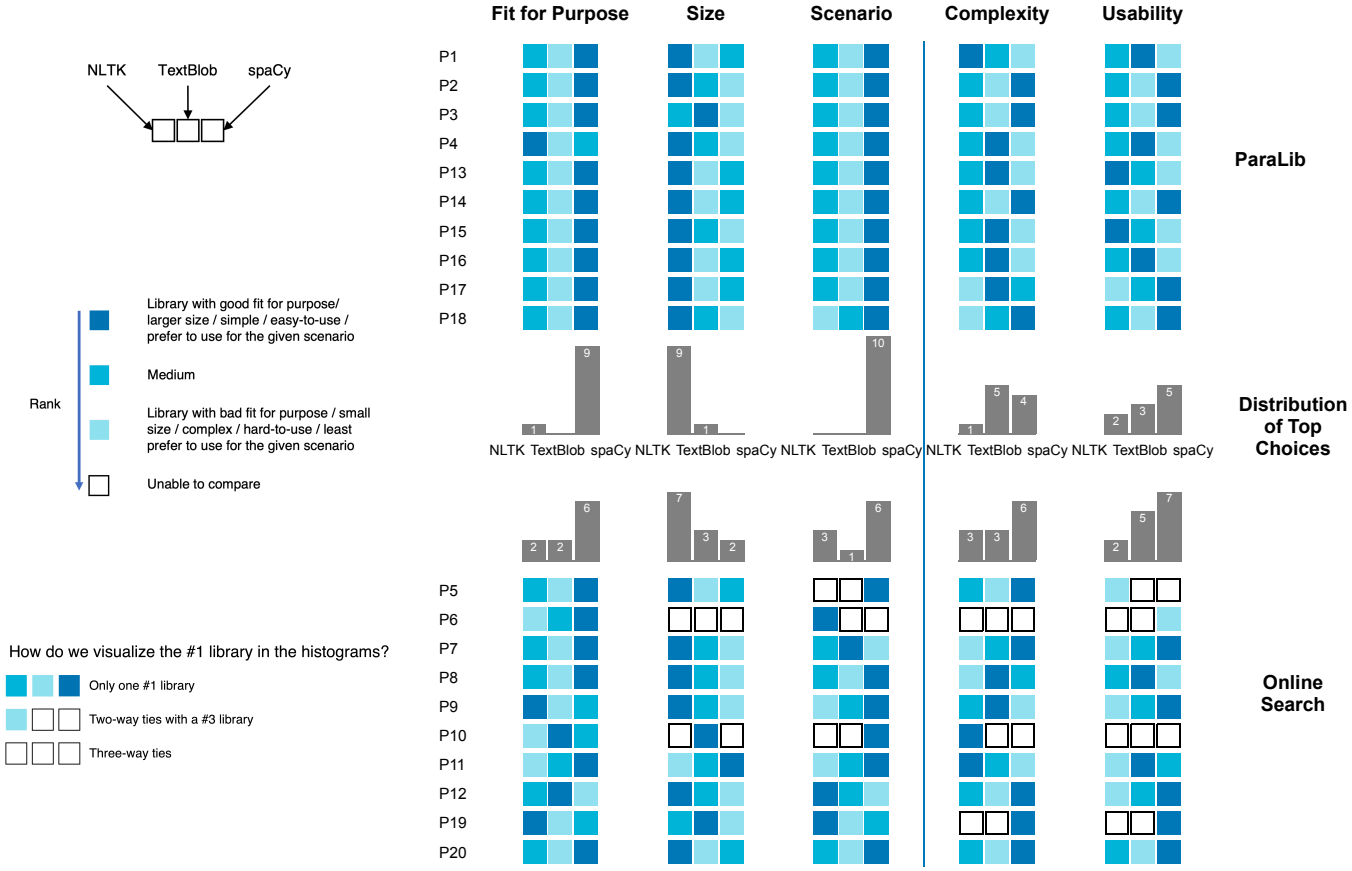


**Figure 4: Twenty participants' rankings of three NLP libraries (`NLTK`, `TextBlob`, and `spaCy`) for fit-for-purpose, code example size, a given scenario, code complexity, and usability, using online search vs.** ParaLib.

three visualization libraries (`D3.js`: $\mu = 1.35, \sigma = 1.79$, `Chart.js`: $\mu = 1.30, \sigma = 1.81$, `Recharts`: $\mu = 0.95, \sigma = 1.70$) and NLP libraries (`NLTK`: $\mu = 1.45, \sigma = 1.88$, `TextBlob`: $\mu = 1.10, \sigma = 1.89$, `spaCy`: $\mu = 1.55, \sigma = 2.09$). In both domains, nearly half of the participants (9 out of 20) had no prior knowledge of using any of these libraries. Four participants rated themselves as somewhat familiar (2–3 on the Likert scale) with using at least one of the libraries in the visualization domain and two in the NLP domain. From the 20 participants we recruited, 3 considered themselves experts (4-5 on the Likert scale) in using at least one of the visualization libraries, and 5 used one of the NLP libraries we mentioned above.

## 7 USER STUDY RESULTS

### 7.1 User Performance

*7.1.1 Participants' library rankings.* Figures 4 and 5 show how participants ranked the suitability of each library along five dimensions. The histograms show how many participants ranked each library as their first choice in each dimension. When using online search, several participants could not make a decision about which library was better in certain dimensions. These are depicted as two-way or three-way ties in white boxes.

In both domains, participants made more consistent library selections when using ParaLib than when using online search. As shown in Figure 4, in the NLP domain, when participants were using ParaLib, 9 out of 10 participants ranked `spaCy` as the top choice in terms of fit for purpose, 9 out of 10 ranked `NLTK` as the top choice in terms of code example size, and all 10 participants ranked `spaCy` as the top choice for the given scenario. Participants' choices were less convergent in terms of complexity and usability. The results are similar for the Visualization domain, as shown in Figure 5.

Participants using online search made more divergent choices and became indecisive when comparing libraries along some dimensions. In the NLP domain, when using online search, only 6 participants ranked `spaCy` as the top choice in terms of fit for purpose, 2 participants chose `NTLK`, and 2 chose `TextBlob`. Similar divergences existed along the other four dimensions. In particular, 4 out of 10 participants could not decide which library was better in at least one dimension when using online search. By contrast, none of them became indecisive when using ParaLib. Compared

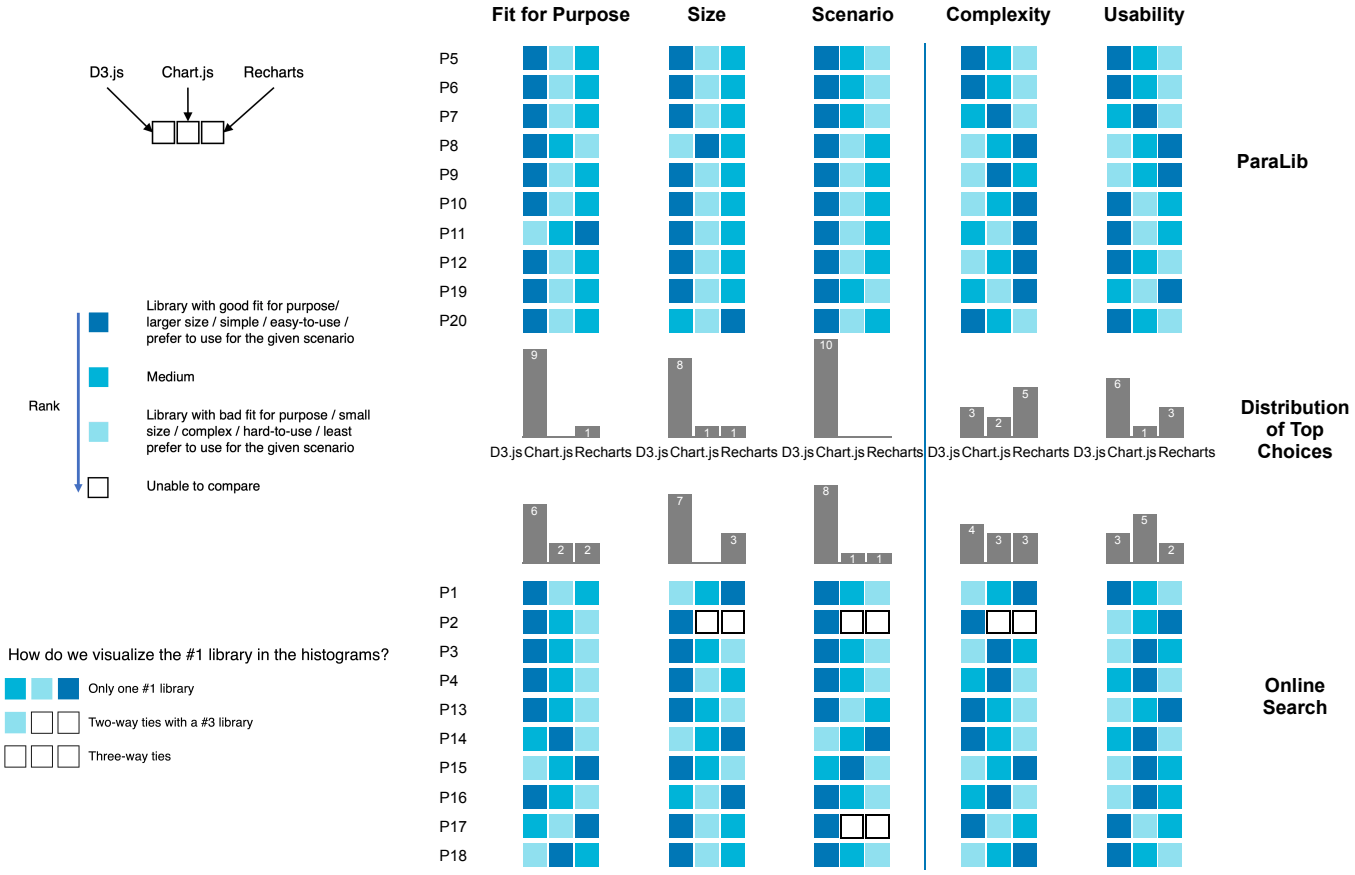


**Figure 5: Twenty participants' rankings of three visualization libraries (`D3.js`, `Chart.js`, and `Recharts`) for fit-for-purpose, code example size, a given scenario, code complexity, and usability, using online search vs.** ParaLib.

with the NLP domain, fewer participants were indecisive in the Visualization domain when using online search, but participants' choices were still more divergent than when using ParaLib.

All 10 participants made the right choice of library for the scenarios in both domains when using ParaLib. In contrast, 4 participants made the wrong choice in the NLP domain and 2 participants made the wrong choice in the Visualization domain when using online search.

In the NLP and Visualization domains, there were 4 and 3 participants, respectively, who were already familiar with one of the three libraries in the set. These participants' library selections indicated some correlation between their prior knowledge and their final choice. For example, P7, who rated himself as an expert (5 on a 6-point Likert scale) in `spaCy`, also ranked `spaCy` as the least complex and easiest-to-use library when using online search. P18, who rated herself as an expert in `D3.js`, also ranked `D3.js` as the easiest-to-use library when using online search. However, without an adequate sample size, we cannot safely draw a conclusion about the relationship between users' prior experience and library selection preferences. We did notice that when using ParaLib, there was no discernible difference in responses between experienced participants (4 or 5 on a 6-point Likert scale) with prior knowledge of at least one library in each domain and participants with much more limited experience (0 or 1 on a 6-point Likert scale).

*7.1.2 Number of code examples inspected during comparison.* By analyzing the study recordings, we manually counted the number of code examples participants inspected in each condition. In both conditions, an example was counted as inspected when participants thought out loud about a code example or selected and highlighted some lines of code when browsing the examples. As results show in Table 1, participants inspected 3 times more examples when using ParaLib than when using online search.

*7.1.3 Participants' responses to the similarities and differences of libraries.* As part of the post-task questionnaire, participants were asked two questions, i.e., (1) describe the similarities and (2) describe the differences, between the set of libraries in each domain. First, we analyzed (1) the length of participants' responses, in characters, with and without counting characters in any code examples within their responses and (2) the number of code examples in their responses. Without removing code examples from our character counts, participants wrote nearly three times longer responses

| | NLTK | | | TextBlob | | | spaCy | | | D3.js | | | Chart.js | | | Recharts | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Median | Max | Min | Median | Max | Min | Median | Max | Min | Median | Max | Min | Median | Max | Min | Median | Max |
| **ParaLib** | 5 | 9.5 | 15 | 4 | 8.5 | 14 | 7 | 10 | 16 | 7 | 11.5 | 15 | 5 | 8 | 14 | 7 | 9.5 | 17 |
| **Online Search** | 0 | 3.5 | 6 | 0 | 2 | 6 | 1 | 2 | 8 | 0 | 3 | 8 | 0 | 3 | 5 | 1 | 3 | 5 |

**Table 1: Statistics about the number of investigated examples using online search vs. ParaLib.**

| | | | Technical Factors | | Specific Functionality | | Human / Social Factors | | Too General or Vague | | Incorrect | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | ParaLib | Online Search | ParaLib | Online Search | ParaLib | Online Search | ParaLib | Online Search | ParaLib | Online Search |
| **Viz** | **Similarities** | **Min** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Median** | 1 | 0 | 3.5 | 0 | 0 | 0 | 0 | 0.5 | 0 | 0 |
| | | **Max** | 5 | 4 | 9 | 5 | 0 | 0 | 1 | 2 | 0 | 1 |
| | | **Mean($\mu$)** | 1.7 | 0.7 | 4.2 | 0.6 | 0 | 0 | 0.1 | 0.6 | 0 | 0.2 |
| | | **$\Delta\mu$** | 1 | | 3.6* | | 0 | | -0.5 | | -0.2 | |
| | **Differences** | **Min** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Median** | 4.5 | 1 | 2.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Max** | 10 | 2 | 4 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |
| | | **Mean($\mu$)** | 4.7 | 1.3 | 2.4 | 0.4 | 0 | 0.4 | 0 | 0 | 0 | 0 |
| | | **$\Delta\mu$** | 3.4* | | 2* | | -0.4 | | 0 | | 0 | |
| **NLP** | **Similarities** | **Min** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Median** | 1.5 | 0 | 3.5 | 1 | 0 | 0 | 0 | 0.5 | 0 | 0 |
| | | **Max** | 4 | 3 | 6 | 5 | 0 | 0 | 1 | 3 | 0 | 1 |
| | | **Mean($\mu$)** | 1.6 | 0.7 | 3.5 | 0.3 | 0 | 0 | 0.1 | 0.7 | 0 | 0.1 |
| | | **$\Delta\mu$** | 0.9 | | 3.2* | | 0 | | -0.6 | | -0.1 | |
| | **Differences** | **Min** | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Median** | 3 | 2.5 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | **Max** | 6 | 5 | 12 | 2 | 0 | 2 | 0 | 2 | 0 | 0 |
| | | **Mean($\mu$)** | 3.3 | 2.1 | 4.1 | 0.7 | 0 | 0.4 | 0 | 0.2 | 0 | 0 |
| | | **$\Delta\mu$** | 1.2 | | 3.4* | | -0.4 | | -0.2 | | 0 | |

**Table 2: Statistics about the number of insights shared by participants in different categories when using ParaLib vs. online search. * indicates statistical significance (paired t-test: $p < 0.05$).**

when using ParaLib than when using online search in both domains. The mean differences in the number of characters between participants' responses in the control and experimental conditions were statistically significant (paired t-test: p-values=0.02398, 0.00970, 0.04367, 0.002494 for both questions in both domains). Even after removing the code examples included in participants' responses, the mean differences in response length were still statistically significant (paired t-test: p-value=0.02983, 0.00911, 0.0014, 0.0022).

The mean differences in the number of examples participants provided between the control and experimental conditions were not statistically significant (paired t-test: p-value=0.16048, 0.06797, 0.33057, 0.16048 for both questions in both domains). This result suggests that participants tended to share more textual description of insights about library similarity and differences when using ParaLib compared with when using online search. This could be attributed to the many concept-annotated code examples shown in ParaLib, through which participants could gather concrete details such as library functionality, syntax, and coding style, and assess potential learnability for them. For example, P5 wrote a detailed comment on the syntax of three visualization libraries: *"Rechart is HTML-based (JSX-based), Chart is CSS-based, and D3 is attribute-based. I personally find Rechart's HTML base very clunky and difficult to keep track of, as HTML syntax can be needlessly pedantic with its various types of brackets (e.g., "{}", "<>"). While D3's attribute-based characteristics are initially off-putting, it is a very clean organization of the parent->child structure—more than the other 2 libraries."*

To better understand the content and quality of participants' responses, the first author manually coded the participants' responses and categorized their insights into five categories:

(1) **Technical Factors** Comments on the technical properties of libraries such as flexibility, learnability, syntax, etc., not including comments on specific functionalities, which has its own category.
(2) **Specific Functionality** Comments on specific functionalities supported by a library (or not).
(3) **Human/Social Factors** Comments on human or social factors, e.g., library popularity, perceived trustworthiness of developers, other users' sentiments, etc.
(4) **Too General or Vague** Comments on some generally known facts about a library, e.g., "*both libraries are written in Python*," or comments that are too vague to understand.
(5) **Incorrect** Comments that contradict statements in official documentation, tutorials, blogs, etc.

Table 2 shows the distribution of different kinds of insights shared by participants when using ParaLib vs. online search. When using ParaLib, participants pointed out significantly more specific functionalities supported or unsupported by a certain library in their responses (paired t-test: $p < 0.05$). Participants also shared

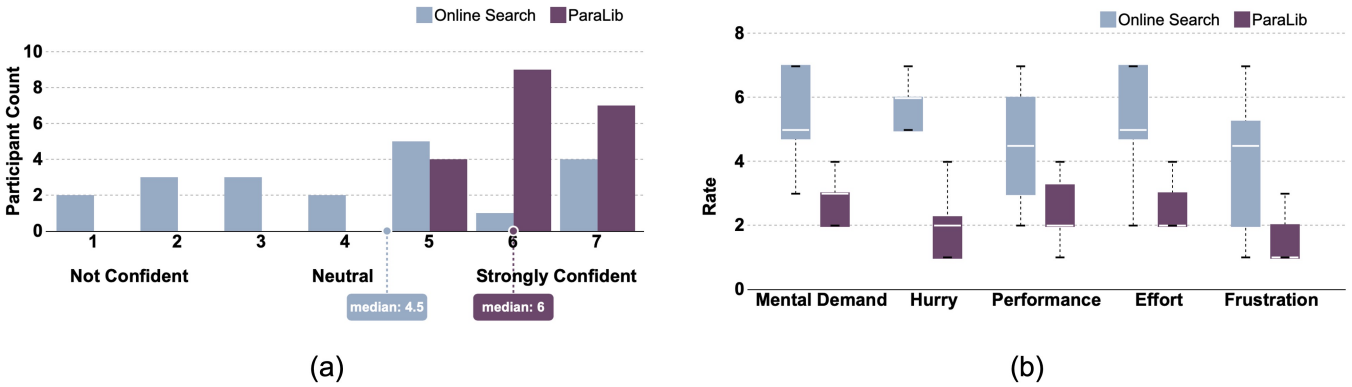


**Figure 6: When using ParaLib: (a) Participants felt more confident in their library selections and comparisons. (b) Participants felt less mental demand, hurry, effort, and frustration during the comparison tasks in both domains, and they reported improved performance when using ParaLib, since a lower score in this figure corresponds to a better perceived performance (paired t-test: $p - value < 0.0001$ in all categories).**

more insights about technical factors and fewer insights about human factors when using ParaLib, though the differences were mostly not significant. Participants shared fewer general or vague insights and made slightly fewer incorrect comments when using ParaLib, though these differences were not significant either. These results suggest that rendering concept-annotated examples in parallel did help programmers obtain a more comprehensive understanding of supported functionalities in similar libraries.

## 7.2 User Confidence and Cognitive Load

Figure 6(a) shows participants' confidence in their library selections and comparisons on a 7-point Likert scale. Visually, it is clear that participants felt more confidence on average when using ParaLib than online search, and the median difference of 1.5 is statistically significant (paired t-test: t=4.1991, df=19, p-value=0.0005). Figure 6(b) shows participants' ratings on the five cognitive factors of the NASA TLX questionnaire. Participants experienced significantly less mental demand, effort, hurry, and frustration when using ParaLib instead of online search. Participants also thought they had better performance (reflected in the figure as a lower score) when using ParaLib. These differences were all statistically significant (paired t-test: $p - value < 0.0001$ in all categories).

Online search provided a less organized plethora of information, with less support for sensemaking and integration than ParaLib. P9 mentioned, *"Using an online search gave me many options, ... I felt that I didn't have enough time to make a great judgment simply because decision-making can be difficult with so many options."* P16 had a similar experience when using online search. He said, *"It's not efficient since so much information will distract me. In most cases, I can't find a direct answer from an online search and I need to integrate this information."* In contrast, P18 said, *"I liked that [ParaLib] provided comprehensive information on three libraries. As it provided example code with the same view, it was easier for me to understand their similarities and differences."*

In the post-study survey, participants directly compared their experiences of using online search and ParaLib (Figure 7). Since we adjusted the post-study survey questions after completing the first 3 studies, there are only 17 responses represented in this figure. As shown in Figure 7(a), 16 of these 17 participants found ParaLib more useful than online search for comparing functionalities across multiple libraries. In Figure 7(b), all 17 participants rated ParaLib easier to use for comparing code examples across multiple libraries. P4 wrote, *"[ParaLib] gave a great visual interface to systematically summarize and compare functionality across different libraries!"* Among the 17 responses, 16 participants felt more confident when using ParaLib in library comparison (Figure 7(c)). Fifteen participants felt less overwhelmed when comparing libraries using ParaLib compared to using online search (Figure 7(d)). Sixteen participants also found ParaLib a helpful resource (Figure 7(e)). They believed ParaLib was not redundant even when online search is available (Figure 7(f)).

Some participants, rather than calling out the novel concept-driven structure provided by ParaLib, ascribed at least some of its value, implicitly, to the fact that it came pre-loaded with a large number of examples that they appreciated and would otherwise have had to go and collect themselves. For example, P19 said, *"There was an overwhelming amount of information (online). Often, I was not able to filter out the necessary information. I was not able to go through a large number of examples as they needed manual searching."*

## 7.3 Qualitative Feedback

The post-task survey of ParaLib asked participants to rate the usefulness of each feature in a 7-point Likert scale. Most participants (19/20) rated the concept hierarchy useful or very useful. Participants mentioned that the concept hierarchy is easy-to-use, good for summarization, detailed, and systematic for comparing multiple libraries in a single view. P19 said, *"I really liked the left panel of the concept hierarchy. It was very detailed. The sub-types [second-level concepts] under each category [first-level concepts] covered a lot of visualization aspects."* P17 mentioned, *"It shows all the available*

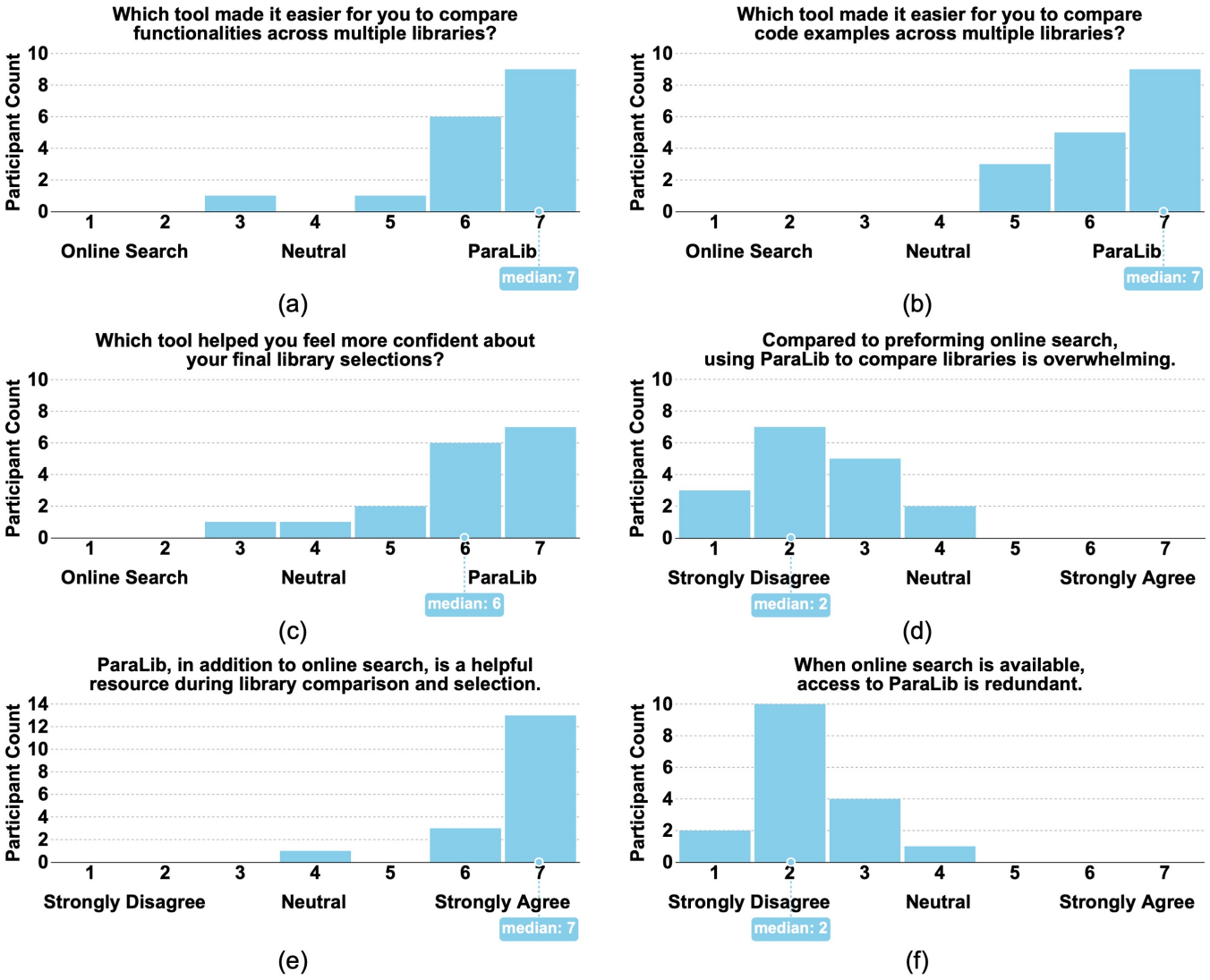


**Figure 7: The distribution of participants' ratings to three comparative questions about ParaLib vs. online search, as well as their agreements to three statements about ParaLib. We adjusted the survey questions after completing 3 studies; therefore, there are only 17 responses in this figure.**

*features of each library in one organized table, which makes it easy to find features."* Seventeen participants also rated the feature of displaying code examples from multiple libraries side by side as more helpful than overwhelming (6 or 7 on a 7-point Likert scale). Other components, such as code examples annotated with concepts (15/20) and highlighting common substrings across multiple code examples (9/20), also received positive reviews. P16 commented, *"Highlights help me quickly find the position of the function and learn how to use the function."*

Participants also commented on the challenges they faced when comparing libraries with online search. Fifteen out of 20 participants complained about the massive amount of information available online, which made the comparison process difficult and overwhelming. P17 said, *"So many resources pop up in the online search results and it's hard to find the one that I really want."* P20 mentioned, *"A lot of information cannot be read at one time. Too much information will leave me unsure of how to judge."* Five out of 20 participants pointed out that online search was time-consuming. Four participants said that the large amount of potentially misleading or not quite comparable information that exists online was distracting. P6 said, *"It was time-consuming (when using online search). I didn't know if I had found the best info for comparison, because I didn't feel like an apples-to-apples comparison."* P4 added, *"I feel that I might*

*have overlooked significant differences in the setup/context needed to use the libraries."*

In both the post-task and post-study surveys, participants made some suggestions to improve ParaLib. Six participants suggested ParaLib include previous programmers' comments, opinions, and reviews. Four participants wanted to have more documentation or tutorials about each library in ParaLib to help them better understand the functionalities and the API usages. Ten participants wanted ParaLib to add more information and measurements to code examples, such as the output of the examples (3/20), the meaning of function calls (2/20), runtime (1/20), library history (1/20), and installation (1/20). Five participants suggested making the mapping from concepts to colors more visually distinguishable.

In the post-study survey, we also asked how their library comparison process would change if they had access to ParaLib in the future. Thirteen participants firmly expressed their willingness to use ParaLib in their future library comparison and selection processes. Among them, 5 people said they plan to use ParaLib in conjunction with an online search to compare libraries. P9 said, *"I'd be able to access code samples and compare them objectively through this site if [ParaLib] had a greater number of samples ... I'd still use it as a supplement to internet searching."* P3 said, *"I will probably first do an online search, and then look at ParaLib to confirm my impression from online search."*

## 8 DISCUSSION

This data suggests that ParaLib clearly fulfilled four of the five interface design goals. Specifically, participants were able to extract usable information from the collection of code examples in ParaLib (**D1**). With this information, participants could assess the relative volume of code necessary to use each library (**D4**) and the functional and syntactic similarities and differences between libraries (**D2**, **D5**). As a result, relative to using existing online resources, participants found it easier to compare and select suitable libraries. They also made more consistent, appropriate library selections, had more insights about specific functionalities, and felt more confident about their decisions.

The study results relevant to ParaLib's utility for usability assessment, e.g., assessing how complex or unfamiliar the syntax is, (**D3**) is less clear, but promising. As mentioned in Section 7.1.1, participants did not readily converge when ranking the complexity and usability of library code examples using either online search or ParaLib. This may be due to participants' prior knowledge and particular preferences. If this is true, this is a positive result: we believe a tool like ParaLib *should* support programmers in making their own personalized assessments along these dimensions. In the open-ended questions on the similarities and differences between the three libraries, half of the participants mentioned that they preferred one library over the other because they felt the library's syntax was more similar to a library they were familiar with. In other words, when confronted with an unfamiliar language or library, they will try to understand it by associating it with a language or library they have used before. For example, many participants mentioned that the syntax of `Recharts` is similar to `HTML/XML/JSX`, which they either did or did not have comfort with. In the NLP domain, P1 mentioned the usage of `spaCy` to train a classifier was similar to training a neural network in TensorFlow. However, the transfer of familiar knowledge to unfamiliar domains may lead to two opposite consequences. On one hand, the knowledge transfer may help them understand the new concept faster by connecting similar familiar syntax, function names, or usages to the unfamiliar code examples. On the other hand, participants may misinterpret information and deviate more from the correct meaning of the new concept on further exploration. For example, after finding the similarities between the three libraries' examples in adjusting the chart's size, P14 mistakenly claimed that all three visualization libraries support canvas.

Future studies with different baselines would be necessary to disambiguate the relative impact of key features of ParaLib's design, e.g., the size of each library's example collection, showing examples from multiple libraries in parallel, annotating code snippets within each example with concepts from a unified concept hierarchy, and showing the distribution over concepts represented in examples in each library's collection. However, the data suggests that, together, they enabled programmers to explore and compare libraries both at a high level, e.g., the general structure and conceptual components of code examples, and a more granular low level, e.g., syntax. And given that many of these key features were possible design implications derived from theories of human concept learning, the success of ParaLib suggests that these theories may be a fruitful source of design inspiration for future designers in this domain.

We described participants' future library selection workflows with ParaLib, and expanding on their answers, we can imagine future tools like ParaLib *supplementing existing online resources* and helping programmers conduct initial investigations, make sense of how one or more libraries are typically used in the wild, compare and select libraries, and identify good examples of a chosen library's usage to adapt to their own purposes or program by bricolage—by remixing concept-annotated snippets from a set of chosen examples.

## 9 LIMITATIONS AND FUTURE WORK

There are a number of limitations to what we know so far, given that ParaLib is a technical probe and only one of multiple possible baselines was used in the user study. For example, since there was one concept hierarchy per domain, created by the authors, our work does not capture how the quality of the concept hierarchy and mapping of concepts to code snippets affects the benefits that ParaLib has to offer.

In our user study, we chose online search as the baseline, but other baselines could have been used to learn different information about what does and does not help programmers compare libraries. These alternative baselines include pre-existing comparison tools like Unakite, Crystalline, or Strata; specific library comparison websites such as StackShare, SassHub, LibHunt; and access to the same collection of examples that were loaded into ParaLib but without any of the key features like concept annotations. The online search baseline does include library comparison websites as well as other sources of complementary information such as official documentation and blogs; during the user study, an average of three library comparison websites (median: 1.5) appeared on the first page of participants' Google search results. This indicates that one alternative baseline, i.e., library comparison websites, was

easily accessible to participants within the baseline we chose. Even so, because we chose a less restricted baseline—online search—we cannot necessarily attribute participants' success to key features such as the concept annotations themselves.

ParaLib does not address the need to consider and compare human factors, e.g., the popularity of a library for a specific task and other programmers' reviews. ParaLib also does not address the need to compare other technical factors, such as maintenance expectations, maturity, stability, and the quality of documentation. Online search can support the comparison of these many other factors—until there are specialized tools to support one or more of these remaining factors more systematically.

Finally, the current process of creating concept hierarchies and concept-annotated example collections is manual, as described in Appendix A. Given the results of the study, we suggest the development of more automated authoring tools for collecting and labeling code examples with concepts. We believe there are two major features worth including in such an authoring tool. First, it should help domain experts create a (possibly evolving) concept hierarchy based on their prior knowledge and existing code examples. Second, it should accelerate the tedious process of annotating code with the concepts in the hierarchy. One possible method would be to ask expert(s) to hand-label a small number of examples, from which the authoring tool could identify similar code patterns, function calls, and comments in yet-to-be annotated examples using machine learning, NLP, or program synthesis methods. The expert(s) could then iteratively review and approve or modify pending propagated concept annotations.

## ACKNOWLEDGMENTS

We would like to thank the 20 anonymous participants in the user study and anonymous reviewers for their valuable feedback. This work was partially funded by NSF grants IIS-1955699, IIS-1955394, and IIS-1956322.

## REFERENCES

[1] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. 2017. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 385–395.

[2] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. 2017. Why Do Developers Use Trivial Packages? An Empirical Case Study on Npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 385–395. https://doi.org/10.1145/3106237.3106267

[3] Maryam Arab, Jenny Liang, Yang Yoo, Amy J Ko, and Thomas D LaToza. 2021. HowToo: A Platform for Sharing, Finding, and Using Programming Strategies. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–9.

[4] Steven Bird, Edward Loper, Ewan Klein, and community. 2001. *NLTK: Natural Language Toolkit*. https://www.nltk.org

[5] Mike Bostock, Jason Davies, Jeffrey Heer, Vadim Ogievetsky, and community. 2011. *D3.js - Data-Driven Documents*. https://d3js.org

[6] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1589–1598. https://doi.org/10.1145/1518701.1518944

[7] Stan Bright. 2014. *LibHunt - Trending open-source projects and their alternatives*. https://www.libhunt.com

[8] Joseph Chee Chang, Nathan Hahn, and Aniket Kittur. 2020. Mesh: Scaffolding Comparison Tables for Online Decision Making. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 391–405.

[9] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2019. What's Spain's Paris? Mining Analogical Libraries from Q&A Discussions. *Empirical Softw. Engg.* 24, 3 (jun 2019), 1155–1194. https://doi.org/10.1007/s10664-018-9657-y

[10] Fernando López de la Mora and Sarah Nadi. 2018. An empirical study of metric-based comparisons of software libraries. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. 22–31.

[11] Joachim Eibl. 2014. *KDiff3 - Homepage*. http://kdiff3.sourceforge.net

[12] Rehab El-Hajj and Sarah Nadi. 2020. *LibComp: An IntelliJ Plugin for Comparing Java Libraries*. Association for Computing Machinery, New York, NY, USA, 1591–1595. https://doi.org/10.1145/3368089.3417922

[13] Dedre Gentner. 1987. *Mechanisms of analogical learning*. Technical Report. Illinois Univ at Urbana-Champaign Dept of Computer Science.

[14] Andreas Gizas, Sotiris Christodoulou, and Theodore Papatheodorou. 2012. Comparative Evaluation of Javascript Frameworks. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12 Companion)*. Association for Computing Machinery, New York, NY, USA, 513–514. https://doi.org/10.1145/2187980.2188103

[15] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. 2015. Foobaz: Variable Name Feedback for Student Code at Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 609–617. https://doi.org/10.1145/2807442.2807495

[16] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 1–35.

[17] Elena L. Glassman, Tianyi Zhang, Björn Hartmann, and Miryung Kim. 2018. Visualizing API usage examples at scale. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[18] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[19] Matthew Honnibal and community. 2016. *spaCy: Industrial-Strength Natural Language Processing in Python*. https://spacy.io

[20] André Hora and Marco Tulio Valente. 2015. apiwave: Keeping track of API popularity and migration. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 321–323.

[21] Amber Horvath, Michael Xieyang Liu, River Hendriksen, Connor Shannon, Emma Paterson, Kazi Jawad, Andrew Macvean, and Brad A Myers. 2022. Understanding How Programmers Can Use Annotations on Documentation. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 69, 16 pages. https://doi.org/10.1145/3491102.3502095

[22] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell Them Apart: Distilling Technology Differences from Crowd-Scale Comparison Discussions. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 214–224. https://doi.org/10.1145/3238147.3238208

[23] StackShare Inc. 2014. *D3.js vs Chart.js vs Recharts | What are the differences?* https://stackshare.io/stackups/d3-vs-plotly-js

[24] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing*. 199–206. https://doi.org/10.1109/VLHCC.2004.47

[25] Nicholas Kong, Tovi Grossman, Björn Hartmann, Maneesh Agrawala, and George Fitzmaurice. 2012. Delta: a tool for representing and comparing workflows. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1027–1036.

[26] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. 2013. Webzeitgeist: Design Mining the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 3083–3092. https://doi.org/10.1145/2470654.2466420

[27] Enrique Larios Vargas, Maurício Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. 2020. Selecting third-party libraries: The practitioners' perspective. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 245–256.

[28] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A Myers. 2019. Unakite: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 67–80.

[29] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2021. To Reuse or Not To Reuse? A Framework and System for Evaluating Summarized Knowledge. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 166 (apr 2021), 35 pages.

https://doi.org/10.1145/3449240
[30] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2022. Crystalline: Lowering the Cost for Developers to Collect and Organize Information for Decision Making. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 68, 16 pages. https://doi.org/10.1145/3491102.3501968
[31] Steven Loria and community. 2013. *TextBlob: Simplified Text Processing*. https://textblob.readthedocs.io/en/dev
[32] Fernando López de la Mora and Sarah Nadi. 2018. Which Library Should I Use?: A Metric-Based Comparison of Software Libraries. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. 37–40.
[33] Walid Maalej and Martin P. Robillard. 2013. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering* 39, 9 (2013), 1264–1282. https://doi.org/10.1109/TSE.2013.12
[34] Ference Marton. 2014. *Necessary conditions of learning*. Routledge.
[35] Brad A. Myers and Jeffrey Stylos. 2016. Improving API Usability. *Commun. ACM* 59, 6 (may 2016), 62–69. https://doi.org/10.1145/2896587
[36] Ahmad Nassri. 2020. *So long, and thanks for all the packages!* https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html
[37] Amantia Pano, Daniel Graziotin, and Pekka Abrahamsson. 2016. What leads developers towards the choice of a JavaScript framework? *arXiv preprint arXiv:1605.04303* (2016).
[38] Amantia Pano, Daniel Graziotin, and P. Abrahamsson. 2016. What leads developers towards the choice of a JavaScript framework? *ArXiv* abs/1605.04303 (2016).
[39] Amy Pavel, Floraine Berthouzoz, Björn Hartmann, and Maneesh Agrawala. 2013. Browsing and analyzing the command-level structure of large collections of image manipulation tutorials. *Citeseer, Tech. Rep.* (2013).
[40] Marco Piccioni, Carlo A. Furia, and Bertrand Meyer. 2013. An empirical study of API usability. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 5–14.
[41] Martin P. Robillard. 2009. What makes APIs hard to learn? Answers from developers. *IEEE software* 26, 6 (2009), 27–34.
[42] Martin P. Robillard and Robert DeLine. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (2011), 703–732.
[43] SaaSHub. 2014. *D3.js VS Chart.js - compare differences & reviews?* https://www.saashub.com/compare-d3-js-vs-chart-js
[44] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 191–201. https://doi.org/10.1145/2786805.2786855
[45] B. Shneiderman. 1996. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*. 336–343. https://doi.org/10.1109/VL.1996.545307
[46] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V. Lopes. 2011. How Well Do Search Engines Support Code Retrieval on the Web? *ACM Trans. Softw. Eng. Methodol.* 21, 1, Article 4 (dec 2011), 25 pages. https://doi.org/10.1145/2063239.2063243
[47] Chart.js Team and Contributors. 2014. *Chart.js | Open source HTML5 Charts for your website*. https://www.chartjs.org
[48] Recharts Team. 2018. *Recharts | A composable charting library built on React components*. https://recharts.org
[49] Kyle Thayer, Sarah E. Chasins, and Amy J. Ko. 2021. A Theory of Robust API Knowledge. *ACM Trans. Comput. Educ.* 21, 1, Article 8 (jan 2021), 32 pages. https://doi.org/10.1145/3444945
[50] Gias Uddin, Olga Baysal, Latifa Guerrouj, and Foutse Khomh. 2021. Understanding How and Why Developers Seek and Analyze API-Related Opinions. *IEEE Transactions on Software Engineering* 47, 4 (2021), 694–735. https://doi.org/10.1109/TSE.2019.2903039
[51] Bowen Xu, Le An, Ferdian Thung, Foutse Khomh, and David Lo. 2020. Why Reinventing the Wheels? An Empirical Study on Library Reuse and Re-Implementation. *Empirical Softw. Engg.* 25, 1 (jan 2020), 755–789. https://doi.org/10.1007/s10664-019-09771-0
[52] Litao Yan, Elena L. Glassman, and Tianyi Zhang. 2021. *Visualizing Examples of Deep Neural Networks at Scale*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3411764.3445654
[53] Tianyi Zhang, Björn Hartmann, Miryung Kim, and Elena L Glassman. 2020. Enabling data-driven api design with community usage data: A need-finding study. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

# A APPENDIX: DATA CURATION PROCESS AND RENDERING

The lead author collected code examples manually using a search engine. For example, for the `D3.js` library, they searched for "D3.js" and "examples" on Google and inspected the top 100 search results. If a search result contained code examples, they manually added the examples to the library's data set. They spent an average of 2 hours finding 50 code examples for each library. Afterwards, they removed any repeated code examples and examples with missing parts. This process took, on average, 15 minutes per library.

With these collections of code examples, the lead author used a simple authoring interface to add concept annotations to snippets within each code example. Specifically, they began with an empty list of functional concepts and read through each example for each library in the same domain. For each unannotated line of code, they would look for the most relevant functional concept in the list of functional concepts collected so far. If there was a suitable functional concept for the unannotated code, they annotated the code correspondingly. If there was no suitable functional concept and the functionality was deemed important enough to capture, they would use the corresponding library's documentation to determine the new functional concept's name, write a description, and add it to the list. By continually repeating this process, a complete list of relevant functional concepts was created, and the code examples were labeled simultaneously. The construction and labeling processes took nearly two hours for three libraries in a single domain. The data curator added first-level concepts to group similar, now subordinate, second-level concepts together from the list, and validated the new concept hierarchy with the rest of the author team. They also checked the annotated code examples and corrected any missing or incorrectly annotated examples. The entire validation process took, on average, 1 hour for each set of three libraries in a given domain.

With the concept hierarchy and labeled code examples, the authoring interface generated a JSON file to include all of the information. We built ParaLib with HTML, CSS, and JavaScript. ParaLib's backend can directly read the JSON file and generate the concept hierarchy and labeled code examples on the interface.

# B APPENDIX: USER STUDY QUESTIONNAIRE

We used the following seven questions to explore users' library selections in the user study.

1. **Fit for purpose** Please rate the libraries based on whether they offer a good match between their set of functionalities and the required functionalities needed in the software. (i.e., the number of tasks/functionalities/purposes each library supports)

2. **Size** Please rate the libraries according to the size of their code examples. (i.e., whether APIs in the library require you to write a lot of code to use them)

3. **Complexity** Please rate the libraries according to the complexity of usage. (i.e., whether a library is easy to use).

4. **Usability** Please rank the libraries according to their usability. (e.g., learnability, understandability, readability, simplicity, etc.).

5. **Similarities** Could you list the similarities of the three libraries?

6. **Differences** Could you list some differences of the three libraries?

(NLP Domain) 7. **Fit for the given scenario** You have a large dataset of 10,000 tweets from Twitter. You want to directly use a pre-trained neural network model to classify tweets into two classes: "relevant" and "irrelevant" to COVID-19. How would you rate your preference for these three libraries if you needed to pick one of the three NLP libraries for this classifier? Important functionalities: 1. The library should be able to provide a pre-trained neural network model. 2. The library can help with text classification)

(Visualization Domain) 7. **Fit for the given scenario** If you needed to complete the task of visualizing COVID-19 infection data (unfiltered) on a geo-map, how would you sort these libraries? requirement: 1. a library can filter data; 2. a library can visualize data on a geo-map (without the need for other plugins).