The JavaScript Package Selection Task: A Comparative Experiment Using *ChatGPT*

Hernan C. Vazquez
Faculty of Sciences
UNICEN University
Tandil, Buenos Aires, Argentina
hvazquez@exa.unicen.edu.ar

J. Andres Diaz-Pace
ISISTAN Research Institute
CONICET & UNICEN University
Tandil, Buenos Aires, Argentina
andres.diazpace@isistan.unicen.edu.ar

Antonela Tommasel
ISISTAN Research Institute
CONICET & UNICEN University
Tandil, Buenos Aires, Argentina
antonela.tommasel@isistan.unicen.edu.ar

***Abstract*—When developing *JavaScript (JS)* applications, the assessment and selection of JS packages have become challenging for developers due to the growing number of technology options available. Given a technology need, a common developers' strategy is to query Web repositories via search engines (e.g., NPM, Google) and shortlist candidate JS packages. However, these engines might return a long list of results. Furthermore, these results should be ranked according to the developer's criteria. To address these problems, we developed a recommender system called `AIDT` that assists developers in the package selection task. `AIDT` relies on meta-search and machine learning techniques to infer the relevant packages for a query. An initial evaluation of `AIDT` showed good search effectiveness. Recently, the emergence of `ChatGPT` has opened new opportunities for this kind of assistants, as reported by some experiments. Anyway, human developers should judge whether the recommendations (e.g., JS packages) of these tools are fit to purpose. In this paper, we report on a user study in which we used both `AIDT` and `ChatGPT` on a sample of JS-related queries, compared their results, and also validated them against developers' criteria and expectations for the task. Our initial findings show that `ChatGPT` is not yet on par with `AIDT` or even human efforts for the task at hand, but the model is flexible to be improved and furthermore, it can provide good arguments for its package choices.**

***Index Terms*—Package Selection, JavaScript, Recommender System, GPT Model, User Study.**

I. INTRODUCTION

In software development in general, and JavaScript (JS) applications in particular, the use of libraries and frameworks can greatly improve developers' productivity in terms of accelerating development cycles and delivering value to customers. Nonetheless, choosing (and reusing) a JS package that fulfills the needs of a development task can be a complex decision-making activity for developers. This complexity stems from the large number of technology options available in Web repositories, such as NPM[1] (Node Package Manager) [1]. Thus, JS developers have to regularly search, evaluate and compare several packages for their applications, and keeping up-to-date with technology becomes challenging. This activity can be perceived as "technological fatigue"[2] by developers.

The poor effectiveness of the JS search engines is a contributing factor to technological fatigue, although these engines have been enhanced over the last years. Developers also resort to general-purpose search engines (e.g., Google or Bing) with the hope of having better results. However, the downside of such engines is that they tend to return long lists of documents, and developers have to navigate within each result to find candidate JS packages. This leads to information overloading issues. Furthermore, once developers identify a set of candidate packages, they must analyze each one to decide the best fit for their needs. Normally, this decision is driven by package features, such as popularity in the community, contributors, or number of downloads, among others. Weighting these features for comparison purposes is not straightforward. For instance, NPM uses an AHP (Analytic Hierarchy Process) [2] technique to compare JS packages.

In prior work, we proposed a recommender system called `AIDT`[3] to assist developers in searching and ranking JS packages [3]. Given a developer's query expressing a technological need, the problem consists in returning a ranking of relevant packages that could satisfy the need of the query – we refer to it as the *JS package selection task*. To tackle this problem, `AIDT` first applies a meta-search strategy [4] that combines results from multiple engines. Based on the recovered packages, `AIDT` then ranks them by relevancy by means of a Machine Learning (ML) model, which is built using a pairwise learning-to-rank method [5]. The ML model can infer a package ranking by analyzing features extracted from JS projects on the Web (e.g., GitHub). We evaluated the `AIDT` effectiveness on the NPM repository using a predefined set of queries and a database with 1000 popular projects from GitHub. In these experiments, we obtained an average precision improvement of 20%, and the tool recovered a larger number of relevant packages when compared to NPM. Furthermore, `AIDT` showed the feasibility of using a data-driven strategy, which can "learn" selection criteria from features from the (open-source) JS community.

The emergence of assistive technologies based on Large Language Models (LLMs), such as OpenAI's ChatGPT, GitHub's Copilot or Google's Bard, has brought new opportunities and challenges for the software community, particularly for development-related tasks. Recent (and preliminary)

979-8-3503-1887-6/23/$31.00 ©2023 IEEE

[1] https://www.npmjs.com/

[2] https://medium.com/@ericclemmons/javascript-fatigue-48d4011b6fc4

[3] Spanish acronym for Intelligent Assistant for Technology Decisions.

experiences evidence that these models can help in several tasks, although they have limitations and pitfalls [6]. In particular, `ChatGPT` [7] works as an advanced general-purpose searcher, which can provide recommendations, rankings, and even justify those recommendations [8]. In this context, given our previous experiences with `AIDT`, a natural question arises: *what is the* `ChatGPT` *performance in the package selection task? Can it do better than humans, or than our tool?* Thus, in this paper, we focus on the tool capabilities to assist developers in JS package selection tasks. We performed three experiments, in which we asked a group of JS developers to work with a sample of queries, and compared their results against those produced by `AIDT` and `ChatGPT` for the same queries, so as to assess the pros and cons of both tools in this regard. From a more general perspective, since assistive technologies are here to stay, we also discuss additional concerns for a package recommendation system to be useful to developers (e.g., contextual explanations of the items being recommended), which go beyond the provision of an accurate ranking of technologies.

The rest of the paper is organized into 5 sections as follows. Section 2 motivates the search and ranking of JS packages, and briefly describes the workings of the `AIDT` and `ChatGPT` tools used in our evaluation. In Section 3, we outline the study design, and then present the research questions and main phases of the experimental procedures. Section 4 reports the findings of our evaluation. Section 5 discusses related works, while Section 6 concludes the paper.

II. THE PACKAGE SELECTION TASK

The selection of a software technology influences both the development process and the quality of the final product [9]. The successful application of a given technology, such as a JS package, means that its usage for a task produces a desired objective [10]. This also depends on contextual features, such as alignment between the developer's need and the chosen package, package maintenance support, or license type, among others. For developing `AIDT`, we departed from two ideas. First, the search and comparison of JS packages can take advantage of multiple information sources. Second, existing JS projects can provide useful information about criteria for assessing the relevance of a given package. In the following, we present a scenario of how the selection works in practice.

A. Motivating example

Let us consider a JS developer that needs to extract a barcode from an image to automate a process for processing codes from an image file. This scenario is illustrated in Figure 1. Initially, the developer goes to the NPM package repository and submits the query "extract barcode from image" to a search engine, which returns only the *bytescout*[4] package as output. *Bytescout* is a JS client for a cloud service. When reading about *bytescout*, the developer realizes that it is a paid service and that the JS client is not open-source. Also, when looking at the description, NPM reports that *bytescout* has been downloaded 40 times in the last month, which might indicate that it is not very popular in the JS community. Let us assume that our developer is not convinced by these features, or that they are not aligned with the project needs. However, *bytescout* is the only technology returned by NPM. In this context, several options arise: (i) adopt the package despite disagreeing with its features, (ii) implement a solution for reading barcodes from scratch, (iii) submit a modified query to NPM to get more results, or (iv) rely on other information sources (e.g., Google, NPMSearch, etc.) to find alternative technologies. Let us suppose that our developer picks the third option and re-phrases the query as "barcode reader", which makes NPM return 16 results this time. After inspecting each result, the developer is still unconvinced about using any of those technologies, since they do not seem very popular nor have enough maintenance. The scenario so far shows the current limitations of JS-specific search engines, like NPM.

Let us assume that our developer goes instead for the fourth option and submits the query "extract barcode from image javascript package" to Google. This query returns a list of Web pages that are inspected by the developer to check whether some JS packages are mentioned. In doing so, our developer realizes that a package called *QuaggaJS*[5] is referenced in three results from the top-10 pages of the list. As the developer is not aware of this technology, she goes back to the NPM repository and finds that *QuaggaJS* is more popular than *bytescout*, it is open-source and well-maintained by the community. At this point, our developer can either pick *QuaggaJS* to fulfill the development need, or keep looking for alternative packages. This scenario illustrates the challenge of using general-purpose search engines for JS packages, as well as the issues related to their comparison.

B. The `AIDT` tool

`AIDT` [3] is a recommender system for the JS domain that works in two stages, which are implemented by separate modules: *ST-Retrieval* and *ST-Rank*, as depicted in Figure 2.

1) ST-Retrieval: This module takes a developer's query and returns a list of candidate JS technologies[6] matching the query. The query is written in natural language and specifies a technological requirement (e.g., "extract barcode from image"). The package retrieval is treated with a meta-search strategy [4], in which the original query is sent in parallel to several search engines, each returning an ordered list of items for the query. `AIDT` relies on four engines to broaden the scope of a query: NPM, NPMSearch[7], Google, and Bing[8]. Each engine returns a set of Web pages (or documents) that might have references to zero or more JS packages. The parsing and extraction of these references is seen as a Named Entity Recognition (NER) task [11], which classifies entities found in a given text into predefined categories (e.g.,

---

[4] https://bytescout.com/

[5] https://serratus.github.io/quaggaJS/

[6] For simplicity, the words "package" and "technology" are used interchangeably as synonyms in the paper.

[7] https://npmsearch.com

[8] https://www.bing.com

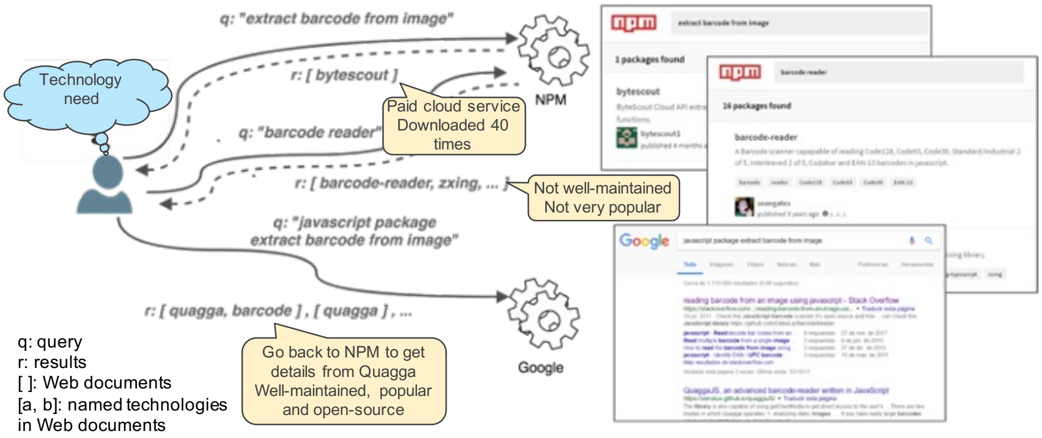


Fig. 1: Example of JS package selection task using different search engines.

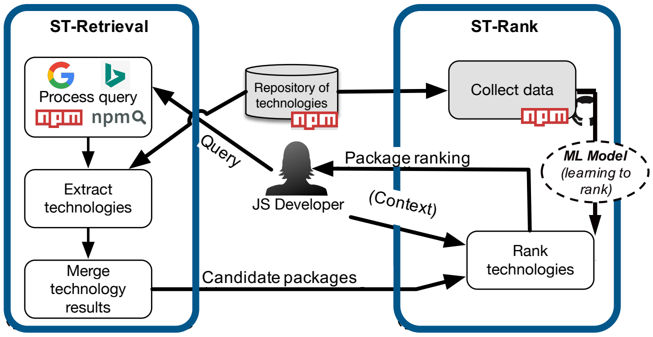


Fig. 2: Overview of the recommendation modules of `AIDT`.

TABLE I: Borda Fuse aggregation example ([points] name).

NPM	Google	Bing	Final list
[4] bytescout	[4] quagga	[4] quagga	[8] quagga
	[3] bcreader	[3] bc-js	[6] bytescout
	[2] bytescout	[2] *bwip-js*	[4] bcreader
	[1] *jaguar*	[1] bcreader	[3] bc-js
			[2] *bwip-js*
			[1] *jaguar*

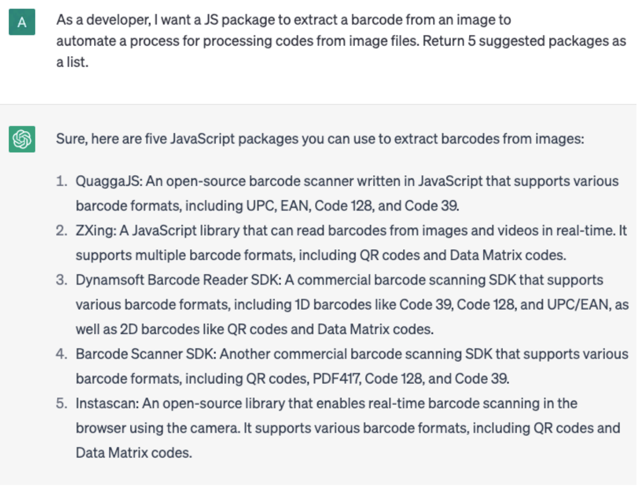
people, organizations, places, among others). In our work, the named entity category is “software technology”, and we apply string-matching rules for identifying named packages. In our example, NPM returned one result (*bytescout*) matching a package name in the repository. Unlike NPM, Google returned an HTML document that was parsed for matches of package names or addresses. In particular, the package name in the resulting page (*QuaggaJS*) did not match any package in the repository (*quagga*), but the site address got a related match (*home_url*, https://serratus.github.io/quaggaJS/).

Based on the named packages extracted, an ordered list of packages per search engine is created. Table I shows an example of the JS packages obtained from the NPM, Google and Bing engines for the query “extract barcode from image”. The individual lists are then combined into a single one using a ranking aggregation function. We rely on the Borda Fuse method for merging the lists [12]. In Borda Fuse, each search engine is considered as a voter with a list of *n* ordered candidates (i.e., the JS packages). For each list, the best first candidate receives *n* points, the second candidate receives *n-1* points, and so on. The points awarded by the different voters are added, and the candidates are ranked in descending order according to the total points obtained. Table I exemplifies a Borda Fuse aggregation for our example, in which the most relevant packages from the individual lists (*quagga* and *bytescout*) ended up at the top of the final list.

2) ST-Rank: This module takes the *ST-Retrieval* output to refine it and produce a better ranking of JS packages. This ranking is constructed by looking at package features and decisions made by other JS projects. This information is crawled from the NPM and GitHub repositories and stored in a technology database. The rationale for incorporating these features into a ranking is that if a package T was selected in a project (over other available options), there should be a criterion that renders T more relevant (than the other options) that is derivable from the features. *ST-Rank* tries to learn this selection criterion by means of a data-driven strategy.

A JS package P is represented by a number of predefined features and its dependencies on other packages. To assemble the dataset, we collected more than 40 features from NPM and Github, including project stars, number of downloads, dependent projects, developers contributing to the project, subscribers, commits, files, or presence of tests, among others. Furthermore, we assess the popularity of a technology T by means of a metric called CDSel (Community Degree of Selection), which models the relationship between the projects in which T was selected and the relevance of those projects. For example, in our database we obtained a CDSel value of 396.192 for *quagga*, 15.646 for *bytescout*, and 1.791 for *bcreader*; which would mean that *quagga* is selected more often than *bytescout* and *bcreader* in the repositories.

The technology database is used for building an ML model to rank JS packages. The training dataset contains a set of training instances, each capturing a pair of technologies and their associated features. Initially, a training ranking is com-



A

As a developer, I want a JS package to extract a barcode from an image to automate a process for processing codes from image files. Return 5 suggested packages as a list.

Sure, here are five JavaScript packages you can use to extract barcodes from images:

1. QuaggaJS: An open-source barcode scanner written in JavaScript that supports various barcode formats, including UPC, EAN, Code 128, and Code 39.
2. ZXing: A JavaScript library that can read barcodes from images and videos in real-time. It supports multiple barcode formats, including QR codes and Data Matrix codes.
3. Dynamsoft Barcode Reader SDK: A commercial barcode scanning SDK that supports various barcode formats, including 1D barcodes like Code 39, Code 128, and UPC/EAN, as well as 2D barcodes like QR codes and Data Matrix codes.
4. Barcode Scanner SDK: Another commercial barcode scanning SDK that supports various barcode formats, including QR codes, PDF417, Code 128, and Code 39.
5. Instascan: An open-source library that enables real-time barcode scanning in the browser using the camera. It supports various barcode formats, including QR codes and Data Matrix codes.

Fig. 3: Example of JS package selection task with `ChatGPT`.

puted for each technology according to its CDSel value. In our example, *quagga* will be ranked first since its CDSel value is higher than those for *bytescout* and *bcreader*. Then, each technology is mapped to a feature vector $[FT_{i1}, FT_{i2}, ..., FT_{in}]$ where FT is an individual feature and n is the total number of features. At last, for each pair T_i and T_j, a pair vector (i.e., a training instance) is created as the concatenation of the feature vectors for T_i and T_j. If T_i is more relevant than T_j, then the label 1 is assigned to the pair, or 0 otherwise. Based on the training dataset, we apply a learning-to-rank technique [5] that works on the instances as if it were a binary supervised classification. The classification model is constructed with the GBRank [13], a popular gradient-boosting algorithm for this purpose. Once built, the ML model can predict the order for any JS package pair, and the results are finally presented to the developer.

C. *The LLM tool:* `ChatGPT`

Over the last few years, LLM technologies [14] have improved the state of the art in several Natural Language Processing (NLP) tasks, by leveraging internal knowledge, and even without relying on external retrievers or further training [15]. In this context, tools like `ChatGPT` [7] provide new means for information seeking, for example, by allowing users to interact with its underlying LLM, ask questions and retrieve information. The emergence of LLMs can be seen as a paradigm shift in research that facilitates in-context learning by simply constructing natural language prompts or instructions [14], [15]. These efforts contribute to further improve the reasoning capabilities and task generalization abilities of LLMs, thereby fostering their application across various domains [8].

Related to our work, LLMs have also created a new paradigm for recommendations based on user instructions [8]. Recommendations can be formulated as prompt-based tasks, where information about users (e.g., developers) and items (e.g., JS packages) gets integrated into personalized prompts as model inputs [8]. For instance, Figure 3 shows a possible prompt and response for the query scenario of Figure 1. It has been argued that LLMs could provide more natural and explainable recommendations, helping to reduce the cold-start problem, and providing cross-domain recommendations [8]. Nonetheless, given that LLMs have not been naturally trained for making recommendations, the quality of the results might suffer [16].

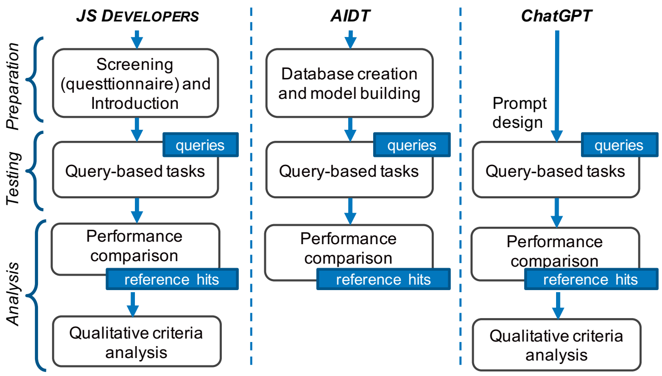


Fig. 4: Experimental design for humans, `AIDT` and `ChatGPT`.

III. STUDY DESIGN

The goals of the study are: i) to assess the feasibility of using assistive tools, like `AIDT` and `ChatGPT`, to deal with the JS package selection task; and ii) to compare their performance against that of human developers in terms of relevance and ranking metrics. In addition to using `ChatGPT`, this study also assesses `AIDT`, because its initial evaluation [3] focused on search effectiveness but the results were not contrasted with human rankings. To this end, we performed three experiments, as depicted in Figure 4.

We addressed the following research questions:

- **RQ#1**: Are the rankings produced by the `AIDT` and `ChatGPT` tools better than those produced by JS developers?
- **RQ#2**: Are there differences in the rankings of `AIDT` and `ChatGPT` tools when compared to the human rankings?
- **RQ#3**: Which selection criteria are considered by JS developers, and the `AIDT` and `ChatGPT` tools?

The experiments involved three phases: preparation, testing and analysis, and were based on the schema of Figure 4. The activities performed at each phase varied depending on the nature of the experiment. For all the experiments, we defined a baseline consisting of set of queries along with reference rankings (of JS packages) for them. NPM was used as the de-facto JS repository. For this baseline, we asked two senior developers to record any queries in NPM that they would make in their projects for a period of two weeks. After filtering out some of their search results (to remove similar queries), we obtained a reference set of 16 queries that represent a variety of technological needs, as listed in Table II. After that, these

TABLE II: Reference queries used in the experiments.

Queries
check valid email address
quick sort algorithm
filter adult content images
user authentication
extract barcode from image
convert typewritten image to text
sentiment analysis
convert text to speech
calculate word similarity
translate english to spanish
credit card validation
captcha authentication
detect text language
DOM manipulation utils
lightweight 3D graphic library
mathematical functions

senior developers provide sets of JS packages, both relevant and non-relevant ones, for the reference queries. The packages in these sets were obtained from NPM and general-purpose search engines. The reasons for including the packages in each set were thoroughly discussed until reaching consensus. Two of the authors participated in the process to refine the results until the reference sets of relevant and non-relevant packages for each query were established.

During the testing phase, each experiment required executing the queries above and collecting the resulting packages. For answering **RQ1#** and **RQ#2**, the performance for the three experiments was evaluated using traditional metrics such as precision, recall, f-measure, MAP and nDCG, based on the reference rankings. The metrics allowed us to assess the relevance of the retrieved packages, but also the effectiveness of the rankings by considering the package positions in such rankings. A cut-off threshold was defined to select the top-k recommended items, where k was set to 5. Thus, all participants, `AIDT` and `ChatGPT` worked with 5 packages for each query.

In this experimental setting, the participants did not interact with the tools (e.g., NPM, Google, Bing, AIDT, `ChatGPT`) directly, but rather the research team did it. This decision tried to reduce the effects of tool learning or UX aspects and focus on the performance of the retrieval task. For instance, the UX design of AIDT is not as intuitive as that of `ChatGPT`. This experimental uniformity had the tradeoff of making the technology selection scenario somehow less realistic.

A. Experiment #1: JS Developers

This experiment was carried out with 21 participants, who were asked to select and assess JS packages for a set of reference queries. These participants were JS developers from a graduate university course (in Argentina) having at least 5 years of development experience. During the preparation phase, we performed an initial screening to determine the participants' level of knowledge and expectations with respect to the study. In addition, each participant received a short introduction to the context of the JS package selection task.

The participants were (randomly) assigned 5 queries each. Internally, we ensured a coverage of all the queries in the set. The queries were phrased as simple as possible. For each assigned query, a participant had to select at least 5 JS packages. In the activity, participants had freedom to choose whatever search engine they judged convenient. They also had to indicate the reasons for selecting and ranking the packages. The maximum time allotted for the activity was 60 minutes. At the end, participants had to complete a post-mortem questionnaire about their level of satisfaction and opinions.

B. Experiment #2: `AIDT`

For configuring and training the *ST-Retrieval* and *ST-Rank* modules, we initially downloaded the package registry from NPM and built a database of technologies up to a given date (September 2017). We ran *ST-Retrieval* 16 times on the reference set (once per query) and stored the aggregated lists of packages. When processing the results, we considered the first 20 documents from the lists of packages, as users searching the Web (e.g., using Google) are very likely to consider only the first pages. We ended up with a total of 2760 JS packages retrieved. The dataset was enriched with package features collected from Github. Furthermore, we relied on NPM and NPMCompare for getting features and alternatives for each package.

For *ST-Rank*, we created a set with ≈ 250 rankings, each one having between $2-6$ packages. In total, more than 1000 training instances were obtained. To validate the rankings produced by the ML model, we split them into training and test sets with the usual $80-20\%$ partition rule of ML tasks. For the test set, we randomly selected a 20% of the training rankings (and their corresponding training instances). These instances were verified by two senior developers. The remaining 80% of the training instances constituted the training set for building the GBRank model. A k-fold cross-validation ($k=5$) was performed to determine the best configuration of hyper-parameters for the model. More details about the construction of the `AIDT` pipeline can be found in [3]. It should be noticed that the process of building the GBRank model behind `AIDT` (particularly, *ST-Rank*) is effort-intensive both in terms of human and computational work. Furthermore, the model should be periodically re-trained with new queries and features from the open-source JS community in order to keep the recommendations current.

Since the ML model of `AIDT` works as a black box regarding the predicted packages, it was not possible to perform a qualitative analysis of the outputs in light of **RQ#3**.

C. Experiment #3: `ChatGPT`

This experiment was based on the `GPT-3.5-turbo model`, which works both in traditional text completion tasks and chat interactions. It allows an input of maximum 4,096 tokens and includes training data up to September 2021. Prompts take the form of a chat conversation in which the user role (i.e., a developer role) asks the model to perform a task (i.e., a technology query). In preparation for the testing phase, after an analysis and refinement of possible prompts, we designed a schema to describe a JS-related requirement and asked `ChatGPT` to recommend 5 packages for the target requirement. We additionally extended the prompt to get a justification of recommendation. In line with **RQ#3**, we intended to understand the selection criteria suggested by `ChatGPT` for each item. To make results comparable and avoid missing relevant packages due to new packages, we included a restriction in the prompt, requiring all recommended packages to have been published before 2018. As for the experiment with JS Developers, we asked `ChatGPT` to provide 3 qualities for each retrieved package in the form of adjectives (to avoid lengthy and non-focused phrases).

Based on the prompt shown in Figure 3, the prompt schema was as follows: "*As a developer, I want a JS package to [extract a barcode from an image to automate a process for processing codes from image files]. Return 5 suggested packages as a list. The packages must have been published before 2018. For each recommended package, include 3 adjectives to justify the choice.*". The part in brackets was iteratively substituted by the different queries in our reference set.

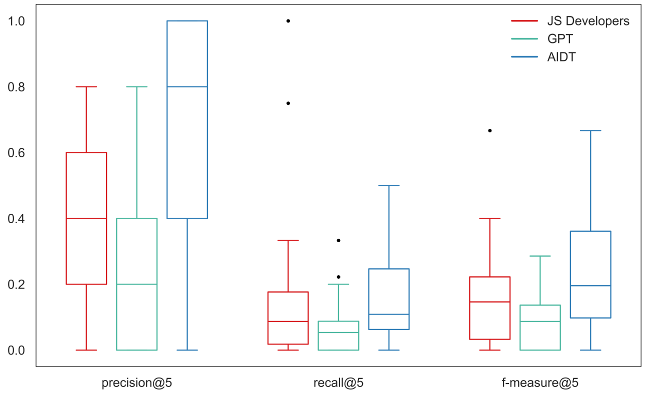
IV. EVALUATION AND FINDINGS

During the analysis phase, we evaluated the results of the three experiments. The analysis focused both on performance aspects (e.g., precision, recall, nDCG, and other metrics) as well as on qualitative ones (e.g., differences between rankings, selection criteria).

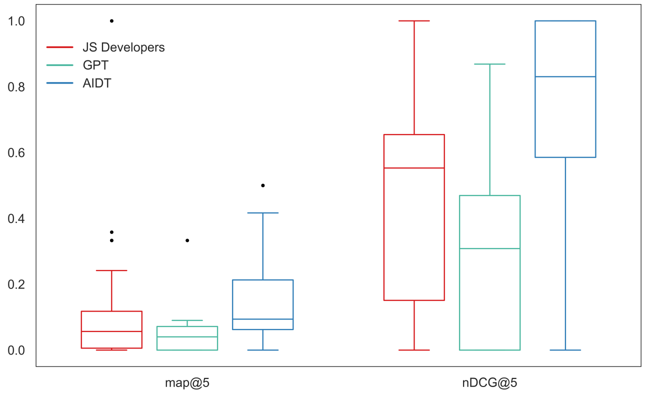
A. *Performance*

The distribution of results for all queries for the package retrieval task are summarized in Figure 5, for the experiments with the JS developers, `ChatGPT`, and `AIDT`. As the figure shows, `AIDT` achieved the best performance, while `ChatGPT` had, in general, lower performance than the human participants. The difference between precision, recall and nDCG indicates that even when recommending non-relevant packages, in general, the relevant ones were ranked high by the three alternatives.

There were queries for which some alternatives did not retrieve any of the relevant packages. For example, developers did not select any of the relevant packages for "lightweight 3D graphic library", `AIDT` did not recommend any relevant package for "translate English to Spanish", "calculate word similarity", "convert typewritten image to text", "filter adult content images" and "lightweight 3D graphic library", while `ChatGPT` was unable to recommend any relevant package for almost half of the queries. Interestingly, `ChatGPT` made recommendations of relevant packages for two of the queries in which `AIDT` failed. This situation might imply that not all queries are easy to satisfy. A manual inspection revealed that in some cases only few packages were relevant (e.g., there was only one relevant package for the "translate English to Spanish" query), which hindered the achievement of that task.



(a) Relevance metrics



(b) Ranking metrics

Fig. 5: Comparison of performance metrics.

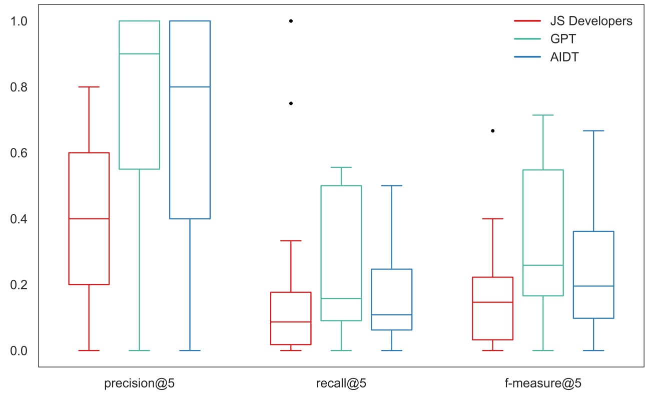
B. *Differences in rankings*

When comparing the rankings from the JS developers and `AIDT`, we observed that `AIDT` achieved better results than the developers, with average improvements of 89% (±194%) and 127% (±246%) for precision and nDCG, respectively. The only exceptions were for the "mathematical functions" and "translate English to Spanish" queries, for which the developers outperformed `AIDT`. Nonetheless, there was a noticeable deviation in the precision and nDCG values for the three alternatives. The largest differences were obtained for the "quick sort algorithm" query. A manual inspection of the rankings defined by the developers revealed non-existing packages (i.e., items that did not belong neither to the relevant nor to the non-relevant sets), whose names partially matched more than one real package. This fact made it difficult to distinguish which packages the developers referred to and, consequently, underestimated their performance.

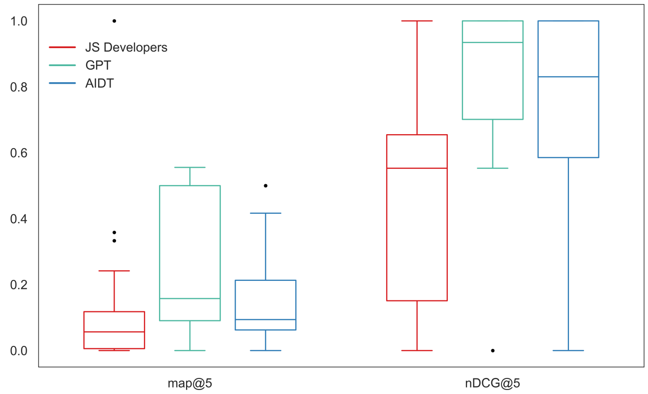
Regarding the rankings for the developers and `ChatGPT`, we noticed that they achieved better results in 12 queries. The largest differences were observed for the "mathematical functions" query. Similarly, when comparing the rankings of `AIDT` and `ChatGPT`, the former outperformed the latter in 10 queries, while achieving the same performance in 4 queries.

In average, `AIDT` had an improvement over `ChatGPT` of 76% (±70%) and 80% (±78%) in terms of precision and nDCG, respectively. The missing packages in `ChatGPT` could be related to the fact that the sources used for training its model could have included a bigger recommendation space than the one analyzed by the JS developers, the experts (for the baseline) or `AIDT`, leading to candidate packages that were unknown to the other parties. There is also a possibility that `ChatGPT` might have returned inaccurate package names or even non-existing packages [16].

Considering the relatively low performance observed for `ChatGPT`, we performed an exploratory evaluation modifying the original prompt, so as to provide a set of candidate packages for each query (i.e., the sets of relevant and non-relevant packages, as defined by the JS experts)[9]. The idea was to help the model to narrow down the search space. With this modification, `ChatGPT` was able to recommend relevant packages for 5 additional queries, improving its average precision and nDCG by 146% (±144%) and 122% (±127%). It is worth noting that for the "lightweight 3D graphic library" query, the model worsened its results and did not recommend any of the relevant packages. In addition, with the modified prompt `ChatGPT` improved its performance with respect to that of the JS developers in 14 queries, and regarding that of `AIDT` in half of the queries. Figure 6 shows the metric values after refining the prompt. This experience highlights the importance of prompt design and how contextual information can positively contribute to the task.



(a) Relevance metrics



(b) Ranking metrics

Fig. 6: Comparison of performance metrics (with the modified prompt for `ChatGPT`).

C. Selection criteria

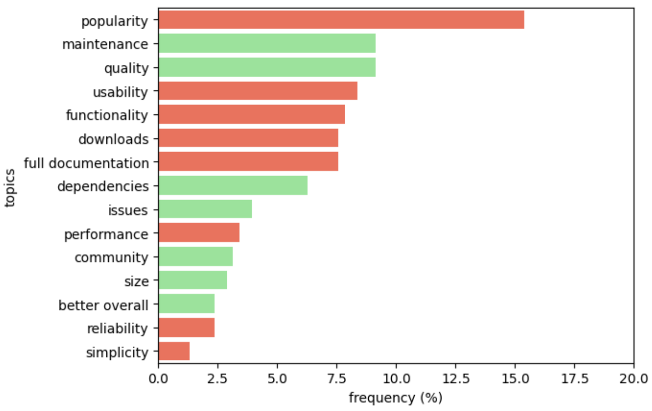
As mentioned in Section III, the JS developers recorded the main aspects they considered for selecting the packages. The prompts for `ChatGPT` also asked for similar information for the returned queries. Figure 7 summarizes the most common criteria in this regard. To facilitate the analysis, we unified these criteria into a common set of topics. The frequent and shared topics by both the developers and *ChatGPT* were related to: "compatibility", "downloads", "efficiency", "full documentation", "functionality", "performance", "popularity", "reliability", "simplicity", "usability", "versatility". From this set, we can see that the topics were related to quality-attribute issues but also to some of the features used by the ML model from `AIDT`. Certain topics (e.g., popularity, maintenance, quality, or usability) had comparatively a higher frequency for the developers ($\approx 10-15\%$) than in the `ChatGPT` responses ($\approx 5\%$), which might indicate a human emphasis, but it does not mean that `ChatGPT` did not consider such criteria.

A further inspection of the developers' responses revealed that not every developer justified every package they chose. Instead, they tended to provide criteria for the first three packages (out of five). In most cases, criteria were expressed using one single word or expression, while only a few developers wrote longer phrases or even paragraphs. For most criteria, it was clear when the developers highlighted a positive aspect of a given package, although the expressions used were ambiguous in some cases. For example, this was the case of the "dependencies" topic, as it was unclear whether developers were referring to packages having a low (i.e., a positive aspect) or high number (i.e., a negative aspect) of dependencies. Things got worse when a developer used that sole criterion for multiple packages.

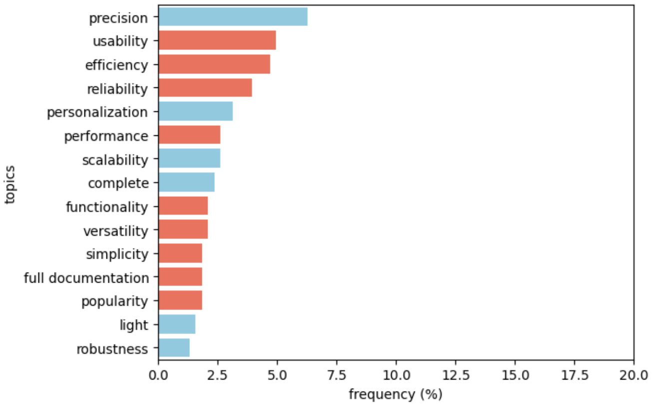
In addition, other developers defined "Best overall" as a criterion without much description of its meaning, which made the topic not comparable to others. We noticed that this subjectivity in the criteria was less apparent in the `ChatGPT` responses. There were subtleties among certain criteria, such as "popularity" and "downloads" or "community". Given that the number of downloads or the size of the supporting community could be indicators of package popularity, their differences were unclear, particularly if used by the same developer. `ChatGPT` also referred to the "popularity of packages", but there was no mention of the number of downloads nor the supporting community.

---

[9] The prompt was modified as follows "*As a developer, I want a JS package to [extract a barcode from an image to automate a process for processing codes from image files]. Given this set of packages [full set of relevant and non-relevant packages], return 5 suggested packages as a list. The packages must have been published before 2018. For each recommended package, include 3 adjectives to justify the choice.*"



(a) Top-15 selection criteria - JS Developers



(b) Top-15 selection criteria - `ChatGPT`

Fig. 7: Main topics identified in package choices. Common topics are highlighted in red. Frequencies are normalized.

The selection criteria given by `ChatGPT`, in turn, were repeated across the recommended packages for only four queries, including repetitions for the first and last packages in the ranking. All topics seemed to respond to positive characteristics of the packages. The causes for the low topic repetition (when compared to the humans' topics) are unclear, and might be due to the prompts used.

Overall, we can answer **RQ#1** by saying that `AIDT` exhibited a superior performance than both the JS developers and `ChatGPT`, particularly in terms of relevant packages. `ChatGPT`, in turn, seems to require more contextual information to produce accurate recommendations. The analysis of the package rankings produced by the different alternatives shed light on their performance, thus answering **RQ#2**. Whereas the behavior of `ChatGPT` has some flexibility, depending on how the prompts are crafted, the behavior of *AIDT* is not changeable once the ML is built. Finally, when addressing **RQ#3**, the analysis of selection criteria for `ChatGPT` and the JS developers revealed more uniformity (i.e., less ambiguity and repetition) in the topics given by `ChatGPT` than in those expressed by humans. Although more evaluation is needed, our results create an opportunity for developers to rely on `ChatGPT` for the package selection task, as a less biased but still informative search engine.

D. Threats to Validity

A number of threats to internal, construct, and external validity were identified in our study, which we tried to mitigate whenever possible.

A first threat, to construct validity, is related to the queries and technology searches used in the experiments. We intended to use queries and search criteria being representative of real-world JS development. The senior developers providing the queries and checking the results might have been biased by the type of software projects they usually work on. Along this line, for `AIDT`, we collected a dataset from the NPM repository using the JS package registry. Despite the low number of queries, 2760 JS packages were returned by the search engines and were manually analyzed. Since analyzing query results might take a substantial amount of time from experts, we preferred not to do a detailed query analysis in this work. To mitigate threats to external validity, we considered queries with different sizes and purposes in the experiments. However, other queries or query phrases for specific domains could have been used. Additional experimentation and surveys with JS developers are still necessary in this regard.

Related to `AIDT`, the usage of *Borda Fuse* in *ST-Retrieval* to rank the package lists is a threat to internal validity, as this method might have biased the results and might have affected the outputs of *ST-Rank* as well. Applying alternative aggregation methods (e.g., Reciprocal Rank, Condorcet) could have generated different package orderings.

In `ChatGPT`, the model could produce multiple responses for the same query. This variation can be related to how the prompt is provided, the training data or even an algorithmic bias. This constitutes a threat to construct validity, as the consistency of the recommendations or the package characterizations (topics) could have been impacted. A potential mitigation is to refine the initial query and carry out an iterative dialogue to ensure consistent responses. We partially showed how the prompts can be refined (to get a better performance (e.g., in terms of relevance or quality) in the results provided by the `ChatGPT` model), but we did not investigate the problem systematically.

V. RELATED WORK

Various techniques have been developed to help select software technologies [17]. Typically, these techniques involve compiling a list of technologies, comparing them, and presenting a ranking to decision-makers. Certain studies have focused on appraising pre-existing technologies but have disregarded searching and acquiring technologies from (Web) repositories. For instance, Ernst et al. [18] proposed a scorecard that assists developers in selecting a particular component from a group of predefined candidate components. This scorecard considers evaluation criteria such as performance, maintenance, and community support.

Software repositories [19] are one of the primary resources for finding technologies. However, current repositories have not been particularly successful in this regard as their search engines often do not provide the desired outcomes. Several studies have attempted to enhance the search mechanisms provided by repositories. In this regard, a few studies share similarities with our approach. Dolphin [20] considers open-source projects, which are ranked based on the extent of their impact (and how frequently they are mentioned) in forum communities, such as StackOverflow or OSChina. The main difference between Dolphin and our approach is that Dolphin only considers open-source projects obtained from version control repositories, while `AIDT` considers software in repositories like NPM. Furthermore, Dolphin does not include general-purpose search engines in the analyses, while `AIDT` includes search results from Bing and Google.

LibFinder [21] employs multi-objective optimization to recommend Java libraries from GitHub and Maven repositories based on source code. Nonetheless, search and recommendation are not guided by user queries. Instead, recommendations are made based on analyzing the source code, aiming at discovering libraries that could replace specific code fragments. As Dolphin, LibFinder does not consider general-purpose search engines. Soliman et al. [22] developed an approach to retrieve architectural decisions and solution alternatives, employing StackOverflow as a repository of architecture knowledge. It is based on a correlation between text (queries) and a "de facto" ontology. Although interesting, the applicability of this approach to JS technologies is still to be demonstrated.

Chen et al. [23] proposed a recommendation technique that relies on a knowledge base extracted from curated Web resources (such as Q&A posts from StackOverflow). Like *ST-Retrieval*, developers' queries are expressed in natural language. Conversely, unlike *ST-Rank*, the approach computes the similarity between the input and the candidate libraries using word embeddings. The information sources examined in this approach can complement the features defined in `AIDT`. Li et al. [24] developed a related approach for searching JS code snippets implementing a particular feature. However, from a development standpoint, it should be noted that reusing snippets is not the same (nor has the same difficulty) as integrating JS packages.

Other works [25] have approached the ranking of technologies according to different criteria. Nonetheless, in most works, the ranking strategies are manually defined based on the features of the candidates. For example, Franch and Carvallo [26] developed a structured quality model for evaluating software packages. This model offers a taxonomy of quality characteristics and metrics for calculating its worth according to the domain at hand. Jadhav et al. [9] used an expert system to combine ranking strategies based on AHP. Instead of following a data-driven strategy, this approach requires experts to define the ranking rules. Finally, Grande et al. [17] conceptualized selection as a multi-objective optimization problem and solved it by means of genetic algorithms.

Reports on using LLMs for software engineering tasks are relatively recent [6]. Assistive tools like `ChatGPT` can provide insights into how developers, users, and stakeholders interact through natural language, leading to enhancements in software development processes and results [27]. For example, `ChatGPT` could be used to identify test cases or test data, explain code fragments or models as a replacement for traditional documentation, or simulate user interactions with software systems to deal with user experience.

Ahmad et al. [28] studied the potential of `ChatGPT` to assist software architects. To this end, the authors presented a case study involving collaborations between architects and `ChatGPT` for the architectural analysis, synthesis, and evaluation of a microservices application. A preliminary evaluation showed that `ChatGPT` was able to imitate the architect's role to support an architecting process by processing user stories, articulating architectural requirements, specifying models, recommending tactics and patterns, and developing scenarios for architecture evaluation. Nonetheless, the experiment still needed a considerable dosage of human oversight and decision support. White et al. [29] also leveraged `ChatGPT` to try to automate common software engineering activities. The authors designed a catalog of prompt patterns covering requirements, system design and simulation, code quality and refactoring tasks. Although this experience is potentially useful, the defined patterns have not been yet validated.

VI. CONCLUSIONS

In this paper, we report on a series of experiments for the JS package selection task, in which we evaluated the results of a group of human developers, against those of the `AIDT` and `ChatGPT` alternatives. Both tools work as recommender systems for assisting developers in selecting, assessing and ranking relevant packages. The tools have differences in their conception. While `AIDT` was explicitly designed for the task, `ChatGPT` is a general-purpose, emerging model that is able to deal with this and other tasks. We performed a comparison using a sample of predefined queries for JS repositories, and then analyzed the rankings returned by each alternative. We did not aim to arrive at statistically significant results from the evaluation, but rather to investigate the pros and cons of the tools and their underlying models.

The results of the experiments so far are enticing. On one side, `AIDT` outperformed both the human developers and `ChatGPT`, particularly in terms of precision. This might be due to the specialized knowledge of the JS domain captured by the ML model. `AIDT` has also shown improvements with respect to traditional search engines. However, `AIDT` is currently unable to explain its rankings, which can compromise the developers' trust on the results. On the other side, `ChatGPT` showed a sub-optimal performance for the task, which seems to be in line with other experiments [6], [16], [28], but it was able to provide good arguments for its selection criteria. Furthermore, its results (and performance, thereof) were affected by how queries are expressed in the prompt. Thus, we observed a tradeoff between having a specialized model (like that of `AIDT`) versus a general-purpose one (like `GPT` or similar LLMs). We believe that, provided with an appropriate configuration (e.g., prompt, tuning), `ChatGPT` could generate more satisfactory package rankings for developers. A related aspect of `ChatGPT` is that its recommendations might change according to the evolution of the available tools, their usage and assessment by JS developers. In fact, for our experiments, we restricted the recommendations up to 2018. This technology evolution can be seen as a concept drift scenario that the recommender system should take into account.

As future work, we plan to extend our study with subjects interacting directly with `ChatGPT`, or other assistants such as Google's Bard, and also allowing newer JS packages as candidates for the queries. In addition, we will further investigate how to incorporate contextual information about the developer's need in the prompts, and also include in the prompts the criteria identified in our qualitative analysis of the experiments (e.g., performance, community support, number of dependencies). A related line of research is an extension of `AIDT` to support explanations of its predictions [30], using the features of its ML model. Another interesting work is the development of a LLM specialization [31] for the JS technology domain [23].

REFERENCES

[1] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the javascript package ecosystem," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 351–361.

[2] T. L. Saaty, "Decision making with the analytic hierarchy process," *International journal of services sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[3] H. C. Vazquez, J. Diaz-Pace, S. A. Vidal, and C. Marcos, "A recommender system for recovering relevant javascript packages from web repositories," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2023, pp. 175–185.

[4] J. A. Aslam and M. Montague, "Models for metasearch," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2001, pp. 276–284.

[5] H. Li, "Learning to rank for information retrieval and natural language processing," *Synthesis Lectures on Human Language Technologies*, vol. 4, no. 1, pp. 1–113, 2011.

[6] B. Combemale, J. Gray, and B. Rumpe, "Chatgpt in software modeling," *Software and Systems Modeling*, May 2023. [Online]. Available: https://doi.org/10.1007/s10270-023-01106-4

[7] M. Abdullah, A. Madain, and Y. Jararweh, "Chatgpt: Fundamentals, applications and social impacts," in *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE, 2022, pp. 1–8.

[8] Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang, "Chat-rec: Towards interactive and explainable llms-augmented recommender system," *arXiv preprint arXiv:2303.14524*, 2023.

[9] A. S. Jadhav and R. M. Sonar, "Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach," *Journal of Systems and Software*, vol. 84, no. 8, pp. 1394–1407, 2011.

[10] A. Birk, "Modelling the application domains of software engineering technologies," in *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*. IEEE, 1997, pp. 291–292.

[11] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 613–622.

[13] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun, "A general boosting method and its application to learning ranking functions for web search," in *Advances in neural information processing systems*, 2008, pp. 1697–1704.

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[15] W. Yu, D. Iter, S. Wang, Y. Xu, M. Ju, S. Sanyal, C. Zhu, M. Zeng, and M. Jiang, "Generate rather than retrieve: Large language models are strong context generators," *arXiv preprint arXiv:2209.10063*, 2022.

[16] Y. Zhang, H. Ding, Z. Shui, Y. Ma, J. Zou, A. Deoras, and H. Wang, "Language models as recommender systems: Evaluations and limitations," 2021.

[17] A. D. S. Grande, R. D. F. Rodrigues, and A. C. Dias-Neto, "A framework to support the selection of software technologies by search-based strategy," in *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*. IEEE, 2014, pp. 979–983.

[18] N. Ernst, R. Kazman, and P. Bianco, "Component comparison, evaluation, and selection: A continuous approach," in *International Conference on Software Architecture Workshops*. IEEE, 2019.

[19] N. Clayton, R. Biddle, and E. Tempero, "A study of usability of web-based software repositories," in *Proceedings International Conference on Software Methods and Tools. SMT 2000*, 2000, pp. 51–58.

[20] Y. Zhan, G. Yin, T. Wang, C. Yang, Z. Li, and H. Wang, "Dolphin: A search engine for oss based on crowd discussions across communities," in *Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on*. IEEE, 2016, pp. 599–605.

[21] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue, "Search-based software library recommendation using multi-objective optimization," *Information and Software Technology*, vol. 83, pp. 55–75, 2017.

[22] M. Soliman, M. Galster, and M. Riebisch, "Developing an ontology for architecture knowledge from developer communities," in *Software Architecture (ICSA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 89–92.

[23] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions – incorporating relational and categorical knowledge into word embedding," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 338–348.

[24] X. Li, Z. Wang, Q. Wang, S. Yan, T. Xie, and H. Mei, "Relationship-aware code search for javascript frameworks," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 690–701.

[25] J. Klein and I. Gorton, "Design assistant for nosql technology selection," in *Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants*. ACM, 2015, pp. 7–12.

[26] X. Franch and J. P. Carvallo, "A quality-model-based approach for describing and evaluating software packages," in *Proceedings IEEE Joint International Conference on Requirements Engineering*. IEEE, 2002, pp. 104–111.

[27] M. A. Akbar and A. A. Khan, "Ethical aspects of chatgpt in software engineering research."

[28] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards human-bot collaborative software architecting with chatgpt," *ArXiv*, vol. abs/2302.14600, 2023.

[29] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," *arXiv preprint arXiv:2303.07839*, 2023.

[30] Y. Zhang and X. Chen, "Explainable recommendation: A survey and new perspectives," *Found. Trends Inf. Retr.*, vol. 14, no. 1, p. 1–101, mar 2020.

[31] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–10. [Online]. Available: https://doi.org/10.1145/3520312.3534862