Unveiling the NPM Ecosystem: Insights into Popularity, Diversity, and Security

V V S Aakash Kotha, Nikita Bhrugumaharshi Emberi

Abstract

The npm ecosystem has grown rapidly, now boasting over 800,000 packages, surpassing other package managers. But understanding its impact goes beyond sheer numbers. This project explores different aspects of the npm ecosystem, like package popularity, gender representation, common domains and keywords, dependency networks, and contributor locations. By analyzing these factors, we aim to uncover trends, community priorities, and potential security issues. Through thorough research and analysis, our project aims to provide valuable insights for developers, researchers, and stakeholders invested in the npm ecosystem's growth and sustainability.

Keywords: npm ecosystem, package popularity, gender representation, development trends, security vulnerabilities, dependency networks, geographic distribution, data analysis.

Our research code and data can be found on (GitHub) for further exploration and validation.

1. Introduction

The npm ecosystem boasts a staggering 230,000 packages, witnessing hundreds of millions of installations weekly [1]. Amidst this abundance, developers encounter a spectrum of resources alongside the challenges of package management. Building upon prior studies examining download trends and GitHub stars [2], our research endeavors to unravel the multifaceted dynamics of package popularity.

In the context of an industry often marked by underrepresentation of women [3, 4], our investigation delves into whether gender diversity receives due attention within the npm package contributor community. Shedding light on gender representation within the npm ecosystem, we aim to highlight areas for improvement & promote inclusivity in open-source development.

Our study aims to investigate gender representation, developer locations, and common themes in npm packages. By identifying trends, we aim to highlight key areas in software development. Our report contributes to understanding npm's landscape and suggests improvements in diversity, security, and community.

Previous analyses [2] have suggested that both highly-selected and not highly-selected packages often serve functionalities related to testing and CSS. While this provides a high-level understanding, our study aims to conduct a more granular examination, uncovering deeper insights into the relationships and dependencies among npm packages. Furthermore, we aim to quantify the prevalence of vulnerabilities using established metrics and assess the vulnerability rate within the npm ecosystem.

Our project aims to fill knowledge gaps and offer practical advice for developers, maintainers, and stakeholders in the software development community. Our main **goals** are:

- Develop effective methods for finding popular npm packages, considering factors like download counts and GitHub stars.
- Analyze gender distribution among npm contributors, aiming to enhance gender diversity for improved team performance.
- Identify main topics and keywords in npm packages, especially popular ones, to understand trends in the ecosystem.
- Create a dependency network using npm package data and Google Pagerank algorithm to identify top packages.
- Investigate global distribution of npm contributors, focusing on popular packages to uncover any geographical disparities.
- Categorize and assess security vulnerabilities in npm packages, particularly popular ones, to address potential risks.

2. Research Questions:

1. Identification of Popular Packages: How can we effectively identify popular npm packages? This entails defining and exploring metrics such as downloads, GitHub stars, and community engagement to gauge package popularity comprehensively.
2. Exploring the gender distribution among contributors in the NPM ecosystem to we investigate potential under representation of females and analyze if popular packages demonstrate a higher proportion of female contributors.
3. Prevalent Domains and Keywords within npm Packages: What are the prevalent domains and keywords within npm packages, particularly among popular ones? How do these patterns reflect current trends and areas of focus within the broader software development community?
4. How does constructing a dependency network from package data in the NPM ecosystem and applying the Google Pagerank algorithm con-

tribute to identifying the top 10 packages, facilitating a deeper understanding and ranking of package importance within the ecosystem?
5. Geographic Distribution of npm Developers: How is the geographic distribution of npm contributors distributed across different countries, especially among popular npm packages?
6. Number and Classification of Vulnerable Packages: What is the number and classification of vulnerable npm packages, particularly among popular ones?

3. Background Work

Mujahid et al.'s [2] analysis of downloads and stars directly addresses how to identify popular packages. This bridges the gap in existing research by specifically focusing on the npm ecosystem and combining developer insights with data analysis to understand why developers choose certain packages. While Mujahid et al. do not directly analyze keywords, their focus on popular packages can be a starting point to explore prevalent domains within those packages.

While existing literature primarily focuses on gender diversity within the broader software development industry, studies by Qiu et al. [4], which include npm developers, and [3] shed light on gender representation among open-source contributors. However, specific analyses of gender representation among npm developers, particularly those contributing to popular packages, are scarce.

Studies by Wang et al [5] and Mujahid et al [2] employ text mining techniques to identify prevalent keywords and domains within npm packages. These studies highlight the significance of understanding package contents and functionalities to discern emerging trends in software development.

The study by Kabir et al [6] offers critical insights into the security landscape of npm packages. By examining vulnerability classifications and identifying contributing factors, this research enhances our understanding of package security. Integrating this perspective into the analysis of popular packages can provide valuable context on the prevalence and nature of vulnerabilities within the npm ecosystem.

While existing research provides foundational insights into various aspects of the npm ecosystem, there remains ample opportunity for further exploration and integration of diverse methodologies to comprehensively understand package popularity, developer demographics, prevalent domains, security implications, and geographic distributions within the npm community.

3. Approach

3.1 Data collection & Data cleaning

3.1.1 Variables considered:

In our research, we carefully selected variables to address each of our six research questions (RQs), ensuring comprehensive coverage and meaningful analysis within the npm ecosystem. Here, we outline the variables considered and their relevance to each RQ:

For RQ1: Identification of Popular Packages:

- **package_name:** This serves as a unique identifier for each package.
- **git_repository:** Provides access to additional properties of the package.
- **stars:** The number of stars on the Git repository, indicating popularity among developers.
- **last_3_month_downloads:** The recent download statistics, reflecting current usage trends.
- **Popularity:** A metric indicating the overall popularity level of the package.

For RQ2: Exploring Gender Distribution among Contributors:

- **contributor:** Unique username identifying contributors.
- **contributor_fullname:** Required for gender prediction, alongside country information, as gender information is not directly available.
- **contributor_country:** Contributor's country of origin, used in gender prediction.

For RQ3: Prevalent Domains and Keywords within npm Packages:

- **keywords:** Reflects the hashtags or keywords associated with each package, crucial for determining prevalent themes and domains.

For RQ4: Dependency Network and Google Pagerank Algorithm:

- **Dependencies_name:** Lists all packages a particular package depends on, essential for constructing the dependency network.

For RQ5: Geographic Distribution of npm Developers:

- **Contributor_country_ISO_code:** Helps in analysing the geographic distribution of contributors across different countries.

For RQ6: Number and Classification of Vulnerable Packages:

- **Health Score:** Indicates the health status of a package, particularly in terms of vulnerabilities.

3.1.2 Dataset Formation:

In our research endeavor, we have structured two distinct datasets to encapsulate the requisite information for our analysis within the npm ecosystem:

Dataset 1: Package Details- This dataset is dedicated to storing comprehensive details about npm

packages, incorporating variables essential for addressing our research questions related to package popularity, vulnerability, dependency networks, prevalent domains, and keywords.

Dataset 2: Contributor Details- This dataset focuses on storing pertinent information about contributors within the npm ecosystem, facilitating analysis regarding gender distribution and geographic representation.

By structuring our datasets in this manner, we ensure clarity and efficiency in handling the vast array of information inherent to the npm ecosystem. Dataset 1 enables us to delve into the characteristics and dynamics of npm packages, while Dataset 2 provides insights into contributor demographics and geographic distribution. Together, these datasets serve as the cornerstone of our research, facilitating rigorous analysis and informed conclusions regarding package popularity, vulnerability, contributor demographics, and more within the npm landscape.

3.1.3 Data Sources Selected:

In our pursuit of robust & comprehensive analysis within the npm ecosystem, we meticulously curated a selection of data sources to procure the requisite info for our research. Each data source was chosen with precision to ensure data integrity, relevance, and reliability. Table 1 outlines the data sources selected for obtaining the variables across both datasets.

3.1.4 Fetching all package_names & filtering subpackages:

In this section, we detail the process of retrieving all package names from the NPM registry and subsequently filtering out subpackages to ensure unbiased analysis. We utilize Python and the requests library to interact with the NPM API.

Retrieving Pkg Names from NPM Registry: We first define a function 'get_npm_packages()' to send a GET request to the NPM API, fetching all package names. This function constructs the request URL and parameters, then extracts package data from the response JSON. The retrieved package names are stored in a list for further processing.

Upon successful retrieval, a total of **2,606,002** package names are fetched from the NPM registry.

Filtering Subpackages: Since subpackages may introduce bias in our analysis, we implement a filtering mechanism to exclude them. We iterate through the list of package names and identify subpackages by checking if the package name contains a forward slash ('/'). An example of a subpackage is '@webjunto/voyage-svg', which is a subpackage of 'webjunto'.

After filtering, only primary packages remain, totaling **1,746,722**.

Through this meticulous process, we ensure that our dataset comprises only primary packages, mitigating any potential bias and facilitating reliable analysis within the npm ecosystem.

3.1.5 Randomly selecting 20k packages:

In streamlining our analysis process due to the time-intensive nature of gathering data for all 1.7 million packages, we opt to randomly select 20,000 packages from the pool of 1,746,722 primary packages. Initially, we import the primary package names from a CSV file and shuffle them to ensure randomness. Subsequently, we select the first 20,000 packages from the shuffled list, creating a representative subset for analysis. These selected package names are then stored in a CSV file for further analysis, providing a manageable yet diverse sample for our research.

3.1.6 Getting NPM Registry API data about 20k packages:

In this section, we describe the process of retrieving and cleaning essential data from the NPM Registry API for 20,000 randomly selected packages. Utilizing Python, Pandas, and the requests library, we interact with the API endpoint (https://registry.npmjs.org/{package_name}) to fetch package metadata, including dependencies, keywords, repository URLs, and download statistics for the last three months as shown in Figure 1.

We begin by reading the list of 20,000 randomly selected package names from a CSV file. For each package, we send requests to the NPM Registry API to fetch metadata, including keywords, dependencies, and repository URLs. Furthermore, we meticulously extract dependencies to ensure accuracy in our analysis. This process involves parsing through package metadata to identify and list dependencies accurately, facilitating comprehensive insights into package interdependencies.

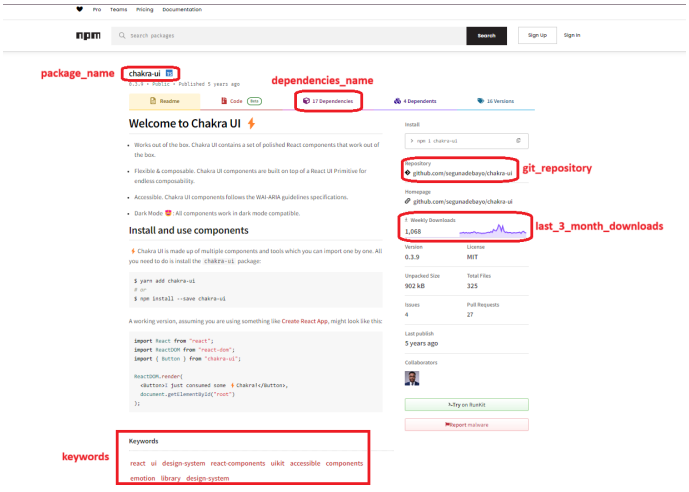

Figure 1: variables fetched from NPM Registry API's endpoint.

Moreover, we aggregate download statistics for the last three months to calculate the total number of downloads accurately. By summing the daily download counts within the specified timeframe, we provide a holistic view of each package's recent usage trends, enabling informed analysis and decision-making.

After fetching the required data, we meticulously clean and organize it into a structured DataFrame. For in-

Data Source	Variables Fetched
NPM registry API [7] (https://registry.npmjs.org/{package_name})	Package_name, git_repository, Last_3_month_downloads, Keywords, Dependencies_name
Git repository of the package	Stars
Synk.io vulnerability page of the package (https://snyk.io/advisor/npm-package/{package_name})	Popularity, Health Score
Git page listing the contributors of the package ({github_repo_url}/graphs/contributors)	Contributor
Git profile page of the contributor (https://github.com{contributors_profile})	Contributor_fullname, Contributor_country

Table 1: Data Sources and Variables Fetched

stance, during the cleaning process, we standardize Git repository URLs to ensure consistency across entries. For example, URLs are converted to HTTPS format, unnecessary prefixes are removed, and any trailing '.git' extensions are eliminated.

Through these rigorous cleaning and organization procedures, we ensure that the data is standardized, accurate, and ready for further analysis, laying the groundwork for insightful exploration of the npm ecosystem.

3.1.7 Getting GitHub data about 20k pkgs:

In this section, we describe the process of fetching Github data, including star counts, for 20,000 selected packages from the npm ecosystem. Leveraging Python and various libraries such as requests, lxml, and git, we interact with Github repositories to gather valuable insights.

We start by reading the previously obtained data for 20,000 packages from a CSV file. For each package, we already extracted the Git repository URL, which serves as the entry point for fetching Github data. We then proceed to fetch star counts (as shown in Figure 2) for each repository by sending HTTP requests to the respective Github pages.

In cleaning the star counts fetched from Github repositories, we adopt straightforward methods to ensure data uniformity and accuracy. If a 'k' suffix is present, indicating thousands (e.g., '3.3k' for 3,300 stars), we multiply the numerical value by 1000 to convert it into an integer representation. We then handle potential errors, such as missing star count elements or unexpected HTML structures, ensuring robustness in our data processing. Finally, we convert the cleaned string representation of star counts into integer values, ready for further analysis and interpretation. After fetching the star counts for each repository, we organize the data into a structured format containing package names, Git repository URLs, and star counts.

Out of the 20,000 packages, Github data, including star counts, is successfully retrieved for 8,080 packages. However, for the remaining 11,920 packages, star counts couldn't be fetched due to various reasons such as improper or broken repository URLs, 404 errors, or repositories located outside of Github. Consequently, our subsequent analysis will be based on the data obtained for these **8,080** packages. Through this comprehensive approach, we gather valuable Github data for a significant subset of npm packages.

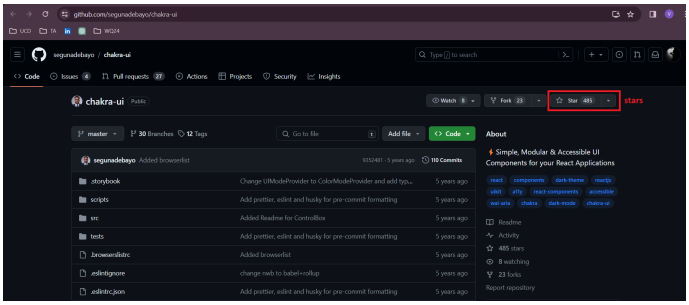

Figure 2: variables fetched from git_repository of every package. Credits

3.1.8 Getting Synk.io vulnerability data about 8080 packages:

In this section, we describe the process of fetching vulnerability data for 8,080 npm packages from Synk.io. Utilizing Python and libraries such as selenium, BeautifulSoup, and csv, we automate the retrieval of vulnerability information for each package.

We begin by reading the package names from a CSV file containing Github data for 8,080 packages. Each package name serves as input for querying vulnerability information from Synk.io. We construct the URL for each package on Synk.io's website (https://snyk.io/advisor/npm-package/{package_name}) and use Selenium with a headless Chrome browser to retrieve the vulnerability data. For each package, we make multiple attempts to fetch vulnerability data, considering potential network issues or webpage loading delays. We extract the health score and popularity metrics (as shown in Figure 3) from the HTML content using BeautifulSoup, ensuring accurate data extraction.

The vulnerability data, including package name, popularity, and health score, is stored in a CSV file named 'synk_data_8k.csv'. This file contains detailed vulnerability information for 8,080 npm packages, enabling comprehensive analysis of package security within the npm ecosystem.

After fetching the vulnerability data, all npmjs, Github, and Synk.io CSV files are merged based on the common attribute "package_name". This merging process results in a unified dataset comprising 8,080 columns, providing comprehensive package details. The final dataset, named "package_details.csv", serves as the foundation for further analysis and insights into the npm ecosystem's security and popularity metrics.

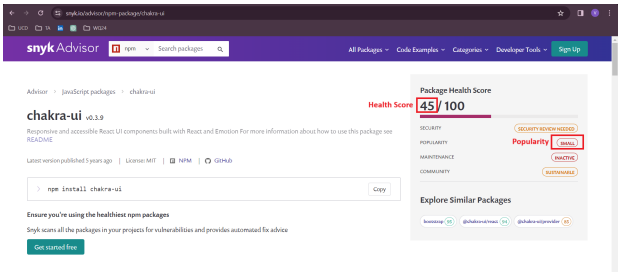

Figure 3: variables fetched from synk.io vulnerability page of every package

3.1.9 Labeling every package into various vulnerability levels & categorizing every package as popular/non-popular:

In this section, we categorize each package into vulnerability levels and label them as popular or unpopular based on predefined criteria. Leveraging Python & Pandas library, we analyze the vulnerability data retrieved from Synk.io and other metrics obtained.

Categorizing Popularity: We begin by determining the popularity status of each package. A package is labeled as popular if it meets certain criteria derived from research [2]. Specifically, a package is considered popular if it falls into one of the following categories:

- It has a popularity (synk.io) other than 'Limited' or 'Small'.
- It has at least 100 stars on Github.
- It has received at least 1000 downloads in the last three months.

Packages meeting any of these conditions are labeled as 'Yes' for popularity; otherwise, they are labeled as 'No'.

Mapping Vulnerability Levels: Next, we map the health score values from Synk.io to vulnerability levels, providing insights into package security. We define vulnerability levels based on thresholds determined from the distribution of health scores observed in the data. Packages are categorized into the following vulnerability levels:

- Very High (Health Score < 40)
- High ($40 \leq$ Health Score < 60)
- Medium ($60 \leq$ Health Score < 75)
- Low (Health Score ≥ 75)

These thresholds are derived from the distribution of health scores observed in the dataset, ensuring that packages are categorized accurately based on their vulnerability severity.

The package data, enriched with popularity labels and vulnerability levels, is stored in a CSV file 'package_details_with_popular_and_vulnerability.csv'. This file serves as valuable resources for further analysis and decision-making regarding package selection and security considerations.

3.1.10 Forming Contributor Dataset:

In this section, we describe our approach to assembling a comprehensive dataset of contributors for GitHub repositories associated with npm packages. We employ various strategies, including web scraping with Selenium, to gather contributor information, including usernames, full names, locations, and predicted genders.

Fetching Contributors' Usernames from Git Repositories:

Initially, we attempted several methods to retrieve contributors' usernames directly from each GitHub repository (as depicted in Figure 4)). These attempts involved scraping the contributors' page URLs ({github_repo_url}/graphs/contributors). However, due to page loading complexities and limitations, these methods were either unsuccessful or too slow.

1. **Attempt-1:** We tried to fetch contributor data directly, but this failed due to incomplete page loading.
2. **Attempt-2:** We utilized Selenium, a web automation tool, but it proved to be too slow for large datasets.
3. **Attempt-3:** We used Selenium with headless mode to improve speed, but the performance remained suboptimal.
4. **Attempt-4:** Further optimizations with Selenium and network conditions were attempted, but the speed was still lacking.
5. **Attempt-5:** Finally, we succeeded by optimizing Selenium with improved network conditions and page load strategies.

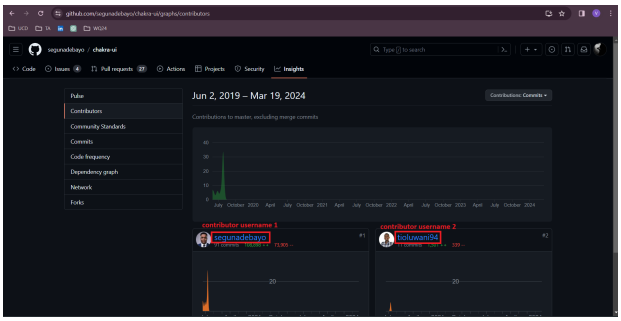

Figure 4: contributor usernames fetched from git_repo of every package. Credits

Fetching Full Names and Locations of Contributors

Subsequently, we extended our approach to gather contributors' full names and locations by delving into individual contributor profile pages on GitHub. Leveraging Selenium and advanced web scraping techniques, we navigated to each contributor's GitHub profile page to extract this crucial information.

Upon reaching each profile page (https://github.com/{contributors_profile}), scraped the unstructured data meticulously, retrieving both the full name and location of the respective contributor (as shown in Figure 5)). However, obtaining the location alone was not sufficient for our analysis. To gain deeper insights and ensure the completeness of our dataset, we sought to augment the location information with additional geographic context. To achieve this, we employed the **Nominatim API**, an unstructured geocoding service.

By submitting the extracted location strings to the Nominatim API, we were able to retrieve detailed geographical information, including the **country code and country name** associated with each contributor's location. This enriched our dataset with standardized and uniform geographical data, facilitating more robust analyses and interpretations.

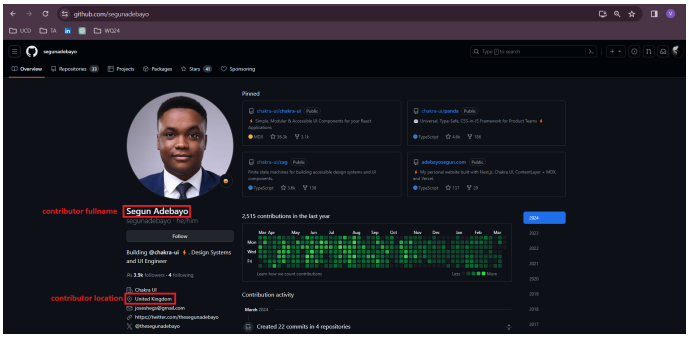

Figure 5: contributor fullname & location fetched from contributor Git profile page from git_repo of every pkg, Credits

Predicting Gender of Contributors

To predict the gender of each contributor, we employed the **GenderComputer** library [8], which utilizes full names and country information to infer gender. This step augmented our dataset with predicted gender labels.

The resulting dataset, named "all_contributors_info.csv," contains comprehensive information about contributors associated with the 8,080 npm packages under study and has **31,763** contributors' info. This dataset serves as a valuable resource for further analysis and exploration of contributor demographics and their impact on package development and maintenance.

3.2 Methodology

3.2.1 RQ-1 Methodology:

To address our first research question on effectively identifying popular npm packages, we embarked on a comprehensive methodology encompassing data collection, metric definition, and exploratory analysis.

In our quest to identify popular npm packages, we rely on three core metrics: downloads, GitHub stars, and package popularity derived from Synk [9] data. Firstly, we scrutinize the download counts of each package, shedding light on its usage and adoption within the ecosystem. Secondly, GitHub stars serve as a testament to community interest and endorsement, providing valuable insights into package popularity. Lastly, we leverage package popularity data obtained from Synk to gauge its relevance and trustworthiness, further enriching our understanding of a package's standing within the npm ecosystem. Together, these metrics offer a holistic perspective on package popularity, facilitating informed decision-making and analysis.

3.2.2 RQ-2 Methodology:

To explore the gender distribution among contributors in the NPM ecosystem, we began by consolidating contributor gender data sourced from individual GitHub profiles. We utilized Selenium and web scraping techniques to extract contributors' full names and locations, subsequently passing this information to the Nominatim API to derive country names and country codes. Following this, we employed the GenderComputer library to predict the gender of each contributor based on their full name and country. The gender prediction enabled us to categorize contributors into male and female genders. Our analysis focused on identifying potential gender underrepresentation and assessing whether popular packages exhibit a higher proportion of female contributors. We divided our analysis into two sections: one exploring gender distribution across all contributors and the other comparing gender ratios between contributors to popular and non-popular packages.

3.2.3 RQ-3 Methodology:

Methodology for keywords:

In order to analyze prevalent domains and keywords within npm packages, particularly among popular ones, we conducted a detailed examination using a multi-step methodology. Initially, we leveraged a dataset containing package details, including information on keywords, obtained from the NPM ecosystem. This dataset underwent preprocessing, including the replacement of NaN values in the "keywords" column with an empty string.

For our analysis, we split the "keywords" string into individual keywords and calculated the frequency of each keyword across all npm packages. This involved iterating over each row in the dataset and incrementing the count for each keyword encountered. Subsequently, we converted the resulting dictionary of keyword counts into a DataFrame and sorted it in descending order based on the count to identify the most common keywords or "hashtags" among npm packages.

Furthermore, to investigate prevalent keywords specif-

ically among popular npm packages, we filtered the dataset to include only packages labeled as "popular." Similar to the previous step, we calculated the frequency of keywords within this subset and generated a DataFrame sorted by count to identify the most common keywords among popular packages.

Methodology for Domains:

```
Topic 0: vue, template, web, generator, framework, sass, css, cli, scss, responsive
Topic 1: image, config, object, array, transform, animation, map, canvas, polyfill, css
Topic 2: cli, npm, nodejs, date, time, node, package, module, console, JavaScript
Topic 3: css, gruntplugin, postcss, data, bootstrap, babel, API, websocket, React, svelte
Topic 4: html, js, testing, typescript, library, graphql, gatsby, minify, translation, parser
Topic 5: aws, utility, mysql, log, google, svg, database, logger, eslint, eslintplugin
Topic 6: api, browser, client, string, parser, parse, random, rest, stream, router
Topic 7: input, simple, event, mobile, select, react, store, queue, hooks, sdk
Topic 8: android, ios, markdown, cordova, promise, crypto, search, documentation, media, plugin
Topic 9: react, component, webpack, plugin, http, validation, json, async, vue, native
Topic 10: javascript, typescript, angular, react, node, eslint, redux, authentication, gulp, auth
Topic 11: test, ethereum, performance, mocha, serverless, utils, reporter, gulpplugin, blockchain, bot
```

Figure 6: Clustered topics and their respective top 10 keywords

Since specific domains are not specified for npm packages, we obtained keywords associated with each npm package from the mentioned NPM API. These keywords were then pre-processed by removing non-alphabetical characters from each one. Subsequently, we transformed this processed keyword list into a bag-of-words corpus, where each document is represented as a list of tuples containing word indices and frequencies. An LDA model with 12 topics was then constructed from this corpus, using the provided dictionary for word-to-ID mapping and iterating over the corpus 15 times for model convergence. The resulting clustered topics and their top 10 keywords are illustrated in Figure 6 Based on these keywords, we labeled the topics as follows:

- **Topic 0: Web Development Frameworks** - Tools for web development like Vue, CSS preprocessors, and responsive design.
- **Topic 1: Data Manipulation and Visualization** - Tools for data handling, transformations, animations, and canvas rendering.
- **Topic 2: Node.js Development Tools** - Tools for Node.js development like CLI utilities and npm package management.
- **Topic 3: Frontend Development Tools** - Tools for frontend tasks like CSS processing and integration with frameworks.
- **Topic 4: Web Development Tools and Utilities** - Tools for HTML, JavaScript testing, and static site generation.
- **Topic 5: Database Management and Utilities** - Tools for database management, logging, and code linting.
- **Topic 6: API Development and Parsing** - Tools for building APIs, parsing data, and routing in web apps.
- **Topic 7: React Development & State Management** - Tools for React UI components and state management.
- **Topic 8: Mobile Development and Utilities** - Tools for mobile app development, encryption, and media handling.
- **Topic 9: Frontend Frameworks and Libraries** - Tools for frontend frameworks like React and Vue.js.
- **Topic 10: JavaScript Frameworks and Tools** - Tools for JavaScript frameworks, code quality checks, and authentication.
- **Topic 11: Testing and Performance Tools** - Tools for testing applications, performance testing, and automation.

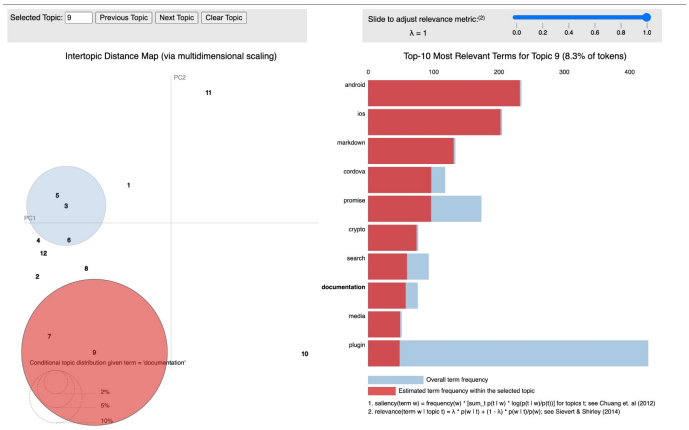

Figure 7: Interactive visualization illustrating the LDA model's findings. The plot displays 10 keywords distributed across 12 topics. Hovering over a topic bubble highlights it in red and shows associated keywords with their probabilities. Hovering over a keyword reveals its potential categories, with bubble size indicating likelihood strength. Click here to interact with the plot.

3.2.4 RQ-4 Methodology:

To identify the top 10 packages within the NPM ecosystem, we employed the Google Pagerank algorithm on a dependency network constructed from the package data. Initially, we extracted package names from the dependencies_name variable and formed a dependency matrix, representing the relationships between packages. This matrix was then converted into a Compressed Sparse Row (CSR) matrix to optimize memory usage due to its sparsity. To handle dangling nodes, where certain packages have no dependencies, we ensured that each row of the matrix sums to a non-zero value by replacing zero rows with equal probabilities for all packages.

Subsequently, we computed the transition matrix 'S' from the normalized matrix 'H', incorporating the damping factor to simulate the random surfer model. After obtaining the Google matrix by combining the damping factor with 'S', we iterated the Pagerank algorithm for 40 iterations to converge to stable rank scores. The final Pagerank scores were utilized to rank all packages, and the top 10 packages were extracted based on their ranks.

3.2.5 RQ-5 Methodology:

To investigate the geographic distribution of npm contributors across different countries, especially among popular npm packages, we utilized data from the "all_contributors_info.csv" file. We first dropped rows

where the "contributor_country_ISO_code" was NaN to ensure data accuracy. Then, we counted the occurrences of each country code in the dataset to determine the country-wise distribution of contributors. Additionally, to specifically analyze contributors associated with popular npm packages, we merged this contributor information with data from "package_details_with_popular_and_vulnerability.csv", filtering only for packages labeled as "popular".

To visualize the geographic distribution, we employed geospatial visualization techniques using the GeoPandas and Folium libraries in Python. This allowed us to create choropleth maps representing the logarithmic counts of contributors in each country, providing insights into the concentration of npm contributors worldwide.

3.2.6 RQ-6 Methodology:

In order to assess the distribution of vulnerability levels among npm packages and investigate their relationship with package popularity, we employed a multi-step approach. Initially, we analyzed the dataset containing package details, including vulnerability information, obtained from the NPM ecosystem. This dataset was preprocessed to extract relevant information, including vulnerability levels categorized as Low, Medium, High, and Very High.

Firstly, to understand the distribution of vulnerability levels among all npm packages, we computed the frequency of each vulnerability level. This involved counting the occurrences of each vulnerability level and calculating the corresponding percentages to provide insights into the prevalence of different levels of vulnerability across the npm ecosystem.

Subsequently, we explored the relationship between package popularity and vulnerability levels. This analysis aimed to determine whether certain vulnerability levels were associated with higher package popularity. For each vulnerability level, we calculated the percentage of popular packages, defined as those receiving significant community attention and usage. This was achieved by identifying the proportion of packages labeled as "popular" within each vulnerability category.

4. Results

RQ-1 Result

Our analysis yielded insightful findings regarding the popularity of npm packages within the ecosystem:

1. **Total Number of Packages**: We analyzed a dataset comprising 8080 npm packages to understand their popularity and distribution.
2. **Popular vs. Non-Popular Packages**: Among the packages studied, 1533 were identified as popular, while 6547 were categorized as non-popular based on our defined criteria.
3. **Percentage Distribution**: Our analysis revealed that 18.97% of the packages were classified as popular, whereas 81.03% fell into the non-popular category, as shown in the Figure 8.

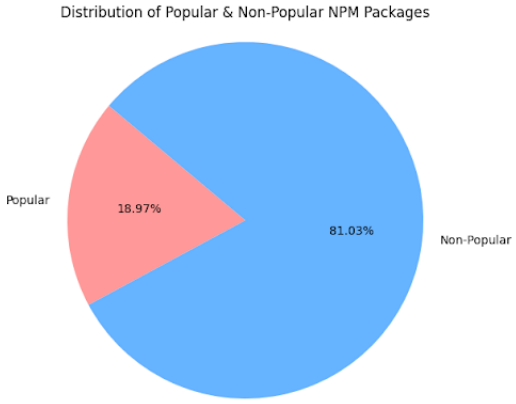

Figure 8: Pie chart showing distribution of Popular & Non-Popular NPM Packages

The majority of npm packages in our dataset are classified as non-popular, indicating a long-tail distribution where a smaller number of packages dominate in popularity. Understanding the factors contributing to package popularity, such as downloads and GitHub stars, is crucial for developers and maintainers seeking to enhance the visibility and adoption of their packages within the npm ecosystem.

RQ-2 Result

Upon analyzing the data, we found that out of the total 31,763 contributors for whom gender data was obtained, **only 8.49%** were identified as female as shown in Figure 9, indicating a significant underrepresentation of females in the NPM ecosystem. This disparity underscores the need for concerted efforts to address gender diversity issues and promote inclusivity within software development communities.

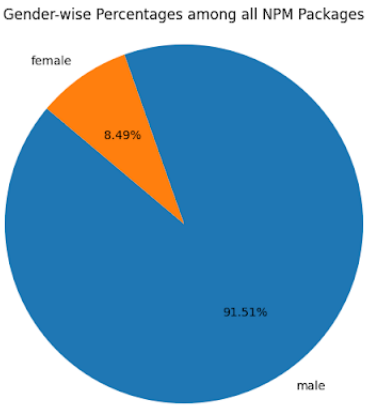

Figure 9: Pie chart showing Gender-wise Percentages among all NPM Packages

Comparing gender distribution between contributors to popular and non-popular packages revealed insightful patterns. Among contributors to popular pack-

ages, the percentage of female contributors was approximately **9.87%**, whereas for non-popular packages, it stood at **6.43%** as depicted in Figure 10. This disparity suggests that popular packages tend to have a higher proportion of female contributors, indicating that diversity within contributor teams may contribute to the overall popularity and success of a package. These findings underscore the importance of fostering diversity and inclusion within software development communities to enhance productivity and innovation.

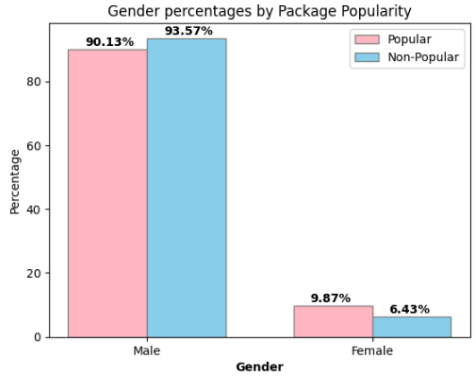

Figure 10: Multi Bar plot showing Gender percentages by Package Popularitye

RQ-3 Result

Result for Keywords:

Our analysis revealed insightful findings regarding prevalent domains and keywords within the npm ecosystem, particularly among popular packages. The top keyword across all npm packages was "react," with a prevalence of 6.34%, indicating its widespread usage and relevance within the npm community. Here, 6.34% for 'react' means that the keyword-'react' is present in the keywords list of 6.34 percent of the NPM Packages, which translates to approximately 164,840 packages (when a total of 2.6M packages is considered), a substantial count within the ecosystem. React emerged as the most frequent keyword in the NPM ecosystem, followed by "javascript" at 2.85% and "typescript" at 2.40% as depicted in Figure 11.

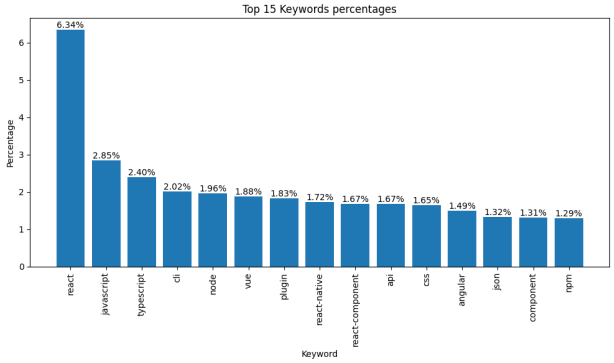

Figure 11: Bar Plot showing Top 15 Keywords percentages

Notably, the keyword "react" maintained its dominance among popular npm packages as well, with a prevalence of 8.94% (as shown in Figure 12), showcasing its significance and popularity among developers. Furthermore, our analysis highlighted the presence of common keywords such as "cli," "node," "vue," and "angular," suggesting the prevalence of frameworks and tools within npm packages. These findings reflect current trends and areas of focus within the broader software development community, emphasizing the popularity of certain technologies and frameworks among developers.

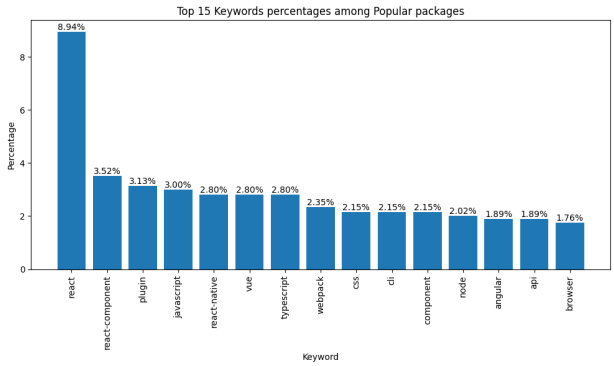

Figure 12: Bar Plot showing Top 15 Keywords percentages among Popular packages

In conclusion, the prevalence of keywords such as "react" underscores the significance of specific technologies and frameworks within the npm ecosystem, influencing package adoption and usage patterns. These insights can inform developers and stakeholders about prevalent domains and technologies within npm packages, facilitating informed decision-making and resource allocation in software development projects.

Result for Domains:

Figure 13: Distribution of domains among all NPM packages

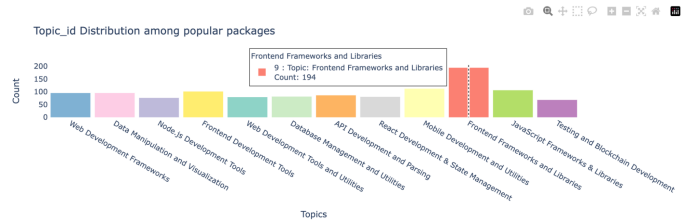

Figure 14: Distribution of domains among popular NPM packages

The LDA model effectively grouped keywords into 12 distinct topics, revealing prevalent themes within npm packages. Figure X visually represents the relative frequency of these topics in the npm ecosystem. Analysis of both all packages and popular ones (Figure 13 and Figure 14) highlights a focus on Frontend frameworks and libraries among developers. Additionally, Mobile Development and Utilities stand out in popular packages, while Frontend Development Tools are prominent overall. Figure 15 supplements this observation by showcasing the usage/popularity levels across different domains, aiding in a deeper understanding of developer preferences and trends.

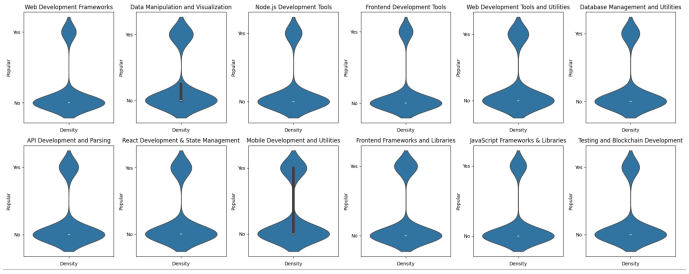

Figure 15: Bean plots depicting the prevalence of topics in popular versus unpopular packages

These findings offer valuable insights for developers, researchers, and stakeholders in the software development community. Understanding the prevalent domains and keywords guides developers in selecting appropriate tools and technologies for their projects, thereby enhancing productivity and project success.

RQ-4 Result

Our analysis yielded the top 10 packages within the NPM ecosystem (our dataset), shown in Figure 16.

Rank	Package Name
1	path
2	json-stable-stringify
3	xmldom
4	modalfy
5	swiff-2
6	npm-author-name
7	vue-smooth-picker
8	input-integer_01
9	har-to-mocks2
10	jquery-github

Figure 16: Bean plots depicting the prevalence of topics in popular versus unpopular packages

These packages, identified through the Pagerank algorithm, represent crucial components within the NPM ecosystem, signifying their significance and influence within the development community. Their consistent appearance in the top ranks highlights their widespread adoption and trust within the community, indicating their reliability and stability as foundational tools relied upon by developers worldwide.

RQ-5 Result

The country-wise distribution of npm contributors revealed that the United States (USA) had the highest number of contributors, comprising approximately 32.41% of all contributors in the dataset. Other countries with significant contributions included Germany (DEU) with 7.01%, the United Kingdom (GBR) with 5.91%, and Canada (CAN) with 5.06% as shown in Figure 17. Notably, these distributions were based on both absolute counts and logarithmic counts, allowing for a comprehensive understanding of the geographic distribution. Click here to view the **Folium interactive map** and explore the distribution across all countries.

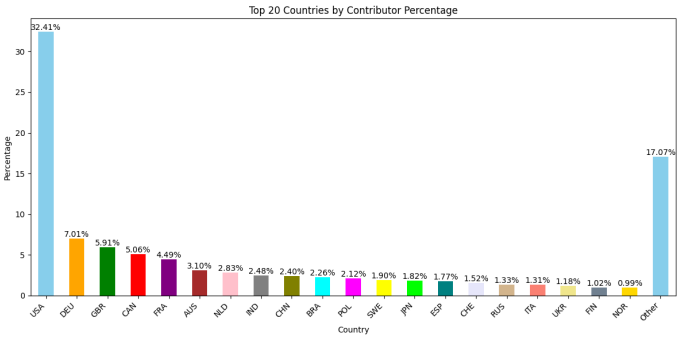

Figure 17: Bar plot showing Top 20 Countries by Contributor Percentage

When specifically considering contributors to popular npm packages, the distribution remained consistent, with the United States (USA) contributing the highest percentage of contributors (approximately 32.27%) as depicted in Figure 18. This suggests that popular npm packages attract contributors from diverse geographic locations, although certain regions, such as North America and Europe, show higher concentrations of contributors. Click here to view the **Folium interactive map** and explore the distribution among contributors to popular packages.

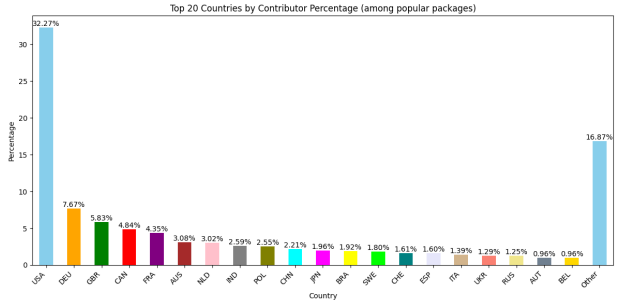

Figure 18: Bar plot showing Top 20 Countries by Contributor Percentage (among popular packages)

The high concentration of contributors from the

United States underscores the significant role of this region in npm package development and highlights the vibrant nature of the software development community in North America. While North America and Europe dominate the contributor landscape, contributions from countries across various continents signify the global reach and inclusivity of the npm ecosystem, facilitating collaboration and knowledge exchange on an international scale.

RQ-6 Result

Our analysis of the distribution of vulnerability levels among all npm packages indicated that the majority of packages (64.98%) were classified as High vulnerability, followed by Very High (30.56%), Medium (3.69%), and Low (0.78%) as shown in Figure 19. This distribution underscores the prevalence of vulnerabilities within the npm ecosystem, highlighting potential areas for improvement in package security and maintenance.

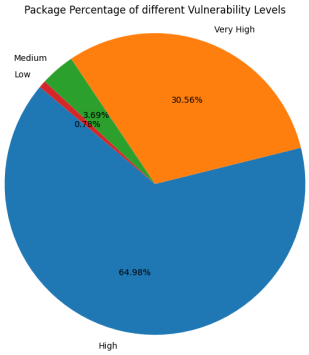

Figure 19: Pie chart showing Package Percentage of different Vulnerability Levels

Furthermore, our investigation into the relationship between package popularity and vulnerability levels revealed intriguing insights. We found that all Low vulnerability packages were classified as popular, indicating that vulnerabilities play a significant role in shaping package popularity. Conversely, only 8.26% of packages categorized as Very High vulnerability were popular, suggesting that severe vulnerabilities might deter users from adopting these packages. Notably, approximately 19.92% of High vulnerability packages were popular (as depicted by Figure 20), indicating that while many individuals utilize these packages, they may require enhanced maintenance and security measures to mitigate associated risks effectively.

These findings underscore the complex interplay between package popularity and vulnerability levels within the npm ecosystem. While vulnerabilities can influence package adoption and usage patterns, effective maintenance practices and security measures are essential to mitigate risks and ensure the reliability of widely utilized npm packages. Further research and proactive measures are warranted to address vulnerabilities effectively and promote the sustainable development of the npm ecosystem.

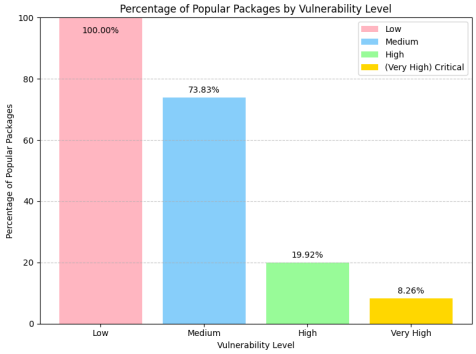

Figure 20: Bar plot showing Percentage of Popular Packages by Vulnerability Level

5. Conclusion

In conclusion, our comprehensive analysis of the npm ecosystem has provided valuable insights into various aspects of package popularity, contributor demographics, prevalent domains and keywords, dependency networks, geographic distribution of contributors, and vulnerability levels. Through our research, we aimed to enhance understanding and inform decision-making within the software development community.

Strengths:

One of the strengths of our research lies in its pioneering nature, as it represents one of the first comprehensive investigations into the dynamics of the npm ecosystem, encompassing diverse research questions and employing advanced methodologies to derive meaningful insights. By exploring multiple dimensions of the npm landscape, we have laid a solid foundation for further studies in this area, contributing to the advancement of knowledge in software development and package management.

Important Results and Inferences:

- Around 19% of npm packages are popular, indicating a long-tail distribution. Understanding factors like downloads and GitHub stars is crucial for package visibility and adoption.
- Female representation in the npm community is low at 8.49%. Popular packages tend to have a slightly higher proportion of female contributors (around 9.87%), hinting at the potential benefits of diversity in contributor teams.
- Keywords like "react" are significant in npm, influencing package adoption. React remains dominant among popular npm packages, reflecting its popularity among developers.
- The LDA model identified 12 distinct topics in npm packages, with frontend frameworks and mobile development standing out in popular packages. This highlights the importance of frontend development tools for developer productivity.
- Contributors to npm come from various countries, with the United States leading at 32.41%. Popular npm packages attract contributors glob-

ally, with North America and Europe showing higher concentrations.

- Most npm packages have high vulnerability levels (about 65%), with very few being low vulnerability. Vulnerabilities impact package popularity, as all low vulnerability packages are popular, while only a small percentage of very high vulnerability packages are popular.

Limitations:

Our research had some limitations. Delays in data collection, mostly due to difficulties with web scraping using Selenium, meant we had to focus on analyzing only 8080 npm packages. While this gave us a detailed look at this subset, our findings might not represent the entire npm ecosystem.

Using publicly available data and predefined criteria to judge package popularity and vulnerability levels could introduce biases. Although LDA helped us find themes in the text data, we need to explore more to make sure it accurately captures the topics.

We also recognize the challenges of getting information from GitHub. Just like with digital trace data, we need to be careful when interpreting it because mining Git repositories, especially on GitHub, is complex. Despite these limitations, they give us chances to improve our research methods in the future.

6. Timeline

Please refer to the Figure 21 for timeline & milestones.

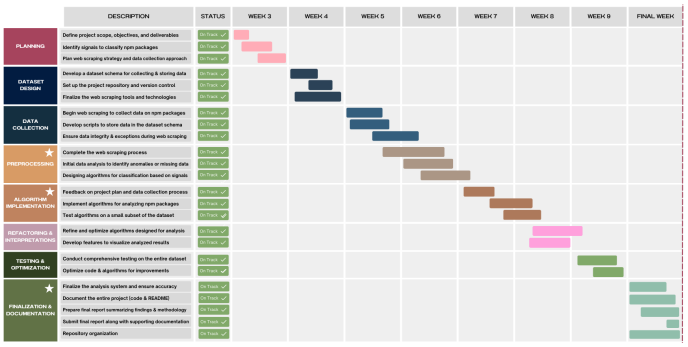

Figure 21: Gantt Chart - Shows the Timeline & Milestones of the project

Acknowledgement

Both of the teammates, V V S Aakash Kotha & Nikita Bhrugumaharshi Emberi, contributed equally to the development of this project. While most of the work was done independently, we utilized Language Models, specifically ChatGPT [10], whenever necessary to confirm details and enhance our understanding. We extend our sincere thanks to it for its valuable assistance throughout the project. Additionally, we would like to thank our **Prof. Peter Kramlinger & TA. Sophia Sun** for their guidance and support. Their insights and feedback were invaluable in shaping the direction of our work.

References

[1] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. A look at the dynamics of the javascript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 351–361, 2016.

[2] Suhaib Mujahid, Rabe Abdalkareem, and Emad Shihab. What are the characteristics of highly-selected packages? a case study on the npm ecosystem. *Journal of Systems and Software*, 198:111588, 2023.

[3] Eitan Frachtenberg and Rhody D Kaner. Underrepresentation of women in computer systems research. *Plos one*, 17(4):e0266439, 2022.

[4] Huilian Sophie Qiu, Zihe H Zhao, Tielin Katy Yu, Justin Wang, Alexander Ma, Hongbo Fang, Laura Dabbish, and Bogdan Vasilescu. Gender representation among contributors to open-source infrastructure: An analysis of 20 package manager ecosystems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 180–187. IEEE, 2023.

[5] Yu Wang, Huaxiao Liu, Shanquan Gao, and Shujia Li. Categorizing npm packages by analyzing the text information in software repositories. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, pages 53–60. IEEE, 2021.

[6] Md Mahir Asef Kabir, Ying Wang, Danfeng Yao, and Na Meng. How do developers follow security-relevant best practices when using npm packages? In *2022 IEEE Secure Development Conference (SecDev)*, pages 77–83. IEEE, 2022.

[7] npmjs all packages. https://github.com/npm/registry/blob/master/docs/REGISTRY-API.md.

[8] Gendercomputer. https://github.com/tue-mdse/genderComputer.

[9] synk.io advisor. https://snyk.io/advisor/.

[10] LLM's : ChatGPT & Gemini