The Emerging Artifacts of Centralized Open-Code

Madiha Zahrah Choksi[*]
mc2376@cornell.edu
Cornell Tech
New York, New York, USA

Ilan Mandel[*]
im334@cornell.edu
Cornell Tech
New York, New York, USA

David Gray Widder
dg536@cornell.edu
Cornell Tech
New York, New York, USA

Yan Shvartzshnaider
yansh@eecs.yorku.ca
York University
Toronto, Ontario, Canada

ABSTRACT

In 2022, generative model based coding assistants became widely available with the public release of GitHub Copilot. Approaches to generative coding are often critiqued within the context of advances in machine learning. We argue that tools such as Copilot are better understood when contextualized against technologies derived from the same communities and datasets. Our work traces the historical and ideological origins of free and open source code and characterizes the process of *centralization*. We examine three case studies —Dependabot, Crater, and Copilot— to compare the engineering, social, and legal qualities of technical artifacts derived from shared community-based labor. Our analysis focuses on the implications these artifacts create for infrastructural dependencies, community adoption, and intellectual property. Reframing generative coding assistants through a set of peer technologies broadens considerations for academics and policymakers beyond machine learning, to include the ways technical artifacts are derived from communities.

CCS CONCEPTS

• **Human-centered computing → Open source software**.

KEYWORDS

Artificial Intelligence, Commons, Ethics, Governance, Free Software, Licenses, Political Economy

ACM Reference Format:
Madiha Zahrah Choksi, Ilan Mandel, David Gray Widder, and Yan Shvartzshnaider. 2024. The Emerging Artifacts of Centralized Open-Code. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24), June 03–06, 2024, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3630106.3659019

[*]Both authors contributed equally to this work.

1 INTRODUCTION

Since its release in 2021, GitHub's programming assistant, *Copilot* has been the subject of extensive commentary and critique. Numerous advocacy groups, such as the Software Freedom Conservancy [94], the Open Source Initiative, and The Free Software Foundation [3], along with scholars from diverse fields such as Machine Learning [7], STEM education [121, 157], HCI [76, 150], and Law [52, 129] have examined the potential benefits, harms and implications of generative coding assistants.

However, the advent of tools such as Copilot was not a given. They rely on the *centralization* of code from the Internet *commons*. Here, we employ a neutral definition of the commons to reflect the open and free access to a resource by all. There are no technical restrictions to free reuse, and open source licenses guarantee that there are no legal restrictions either [23, 65, 117, 137].

We describe a centralizing tendency in the hosting and sharing platforms for Free and Open Source Software (F/OSS). Centralization enables the creation of a new class of tools we term *centralized commons based tooling*. Copilot is one such tool. This work examines the broader category of tools Copilot falls under. Although centralized commons based tools are not necessary for centralization, centralization is, however, necessary to *produce* such tools. Our research shows how the form and function with which centralized commons based tools interact with the technical communities that produce the underlying data exist along spectrum's of social, technological, political and economic valence.

Langdon Winner shows how technical artifacts have politics: he demonstrates how ostensibly neutral tools— from bridges to bombs to tomato harvesters depend on and inscribe relations of power in society [165]. Just as text and vision models are trained on vast corpora of images and text [9] produced on social platforms [140], the code used to train models like Copilot is produced on GitHub.

Package managers and online code hosting simplified collaborative software development [40]. Social features drove developers to share and collaborate on GitHub where open source software became increasingly centralized [50]. The scale of open source development activity drove the use of automation tools for dependency management and continuous integration services [63]. As it became the dominant infrastructure underling the open source ecosystem, GitHub amassed a vast trove of data, beyond code to include dependency graphs, historical commits, user interactions and documentation. In 2018, Microsoft acquired GitHub for $7.5 billion USD, 30 times its annual recurring revenue. The purchase was widely understood as bid for its *strategic* rather than its *financial* value [156]. Three years later, it was the synthesis of GitHub's data, Microsoft's compute resources, and OpenAI's machine learning expertise that produced Copilot [53].

Our contribution is two-fold. First, we present a historiography of how F/OSS software became centralized on platforms and package managers. Building on similar work in critical data studies [93], and historical analyses of technical development [33], we examine the *process* by which F/OSS code became centralized. Second, we juxtapose Copilot with two other examples of centralized commons based tools which have different technical, participatory, and legal implications. We adopt a case-study approach [59] to compare three large-scale tools that rely on centralized open source code: *Crater*, a tool for testing the Rust compiler; *Dependabot*, a system which scans open source software repositories to keep software dependencies up-to-date; and *Copilot,* a code completion subscription product by GitHub.

We explore the relationship of these tools to their infrastructural dependencies, community interactions, and implications for an open commons. In our discussion, we advocate for an expansive perspective of the modern open source ecosystem, to acknowledges the heterogeneity of tools generated from a communal wealth of open source code. We approach this work through interdisciplinary lenses, spanning computer science, history, human computer interaction, law, and software engineering.

2 RELATED WORK

2.1 Situating the Commons

In theorizing about the commons, Elinor Ostrom's foundational work on governance challenges the idea that shared resources, or commons, are necessarily overused and depleted [117]. Open source communities manage *information* common resources. They are dedicated not to the production of tangible goods like crops or oil, as in Ostrom's original examples [117] but information goods that are non-rival and non-excludable. This difference simultaneously creates opportunities (because those goods can be freely shared without depletion) and challenges (because it can be harder to define and enforce communities boundaries and rules) [65]. In addition to common-pool resources, open source communities also create intellectual and social infrastructure [23, 54] to support the community's operations. These blended infrastructures draw on intellectual and cultural resources to showcase a symbiotic relationship between technical innovations and sociocultural contexts.

2.2 Ethics and Openness

Ethical issues in open source software have also been an object of academic scholarship. Gabriella Coleman examines the ethics of open source hacking in the context of Debian, and demonstrates how self-governance within the community critiques and re-invents wider neoliberal ideas of political participation [29]. In her work, Coleman challenges conventional notions of authority around how communities democratize access to knowledge and technology [30]. Innovations in licensing created complex edge cases around fair use in both the legal and ethical sense [66]. Nissenbaum [113] demonstrates how the "many hands" involved in software production make it harder to hold any one person accountable. In contrast Grodzinsky et al. [67] argues that open source allows peers to hold each other accountable for low quality code.

In the context of AI, Widder et al. [161] distinguishes between "implementation" harms, which open source may help prevent through its transparency, and "use" harms that more easily proliferate with open source distribution. Solaiman [138] catalogs middle-ground approaches between closed and open, such as staged release and access via an API. Others suggest behavioral use licensing [31] may help avert some harmful uses.

The many stages of widely adopted, dynamic AI pipelines and multi-purpose toolkits further exacerbate the "many hands" problem [32]. The fractured inter-firm nature of AI supply chains both means developers do not feel accountable for use [160] and introduces an "accountability horizon" where they do not know about how software may be later used [28]. Given this, AI Ethic's focus on documenting bias and producing related metadata has been critiqued in cases where these interventions can be easily lost or ignored as ML artifacts are remixed and reused [55].

2.3 Critical Work on the Power and Politics of Datasets

Datasets are far from neutral, "the work of producing, preserving, and sharing data reshapes the organizational, technological, and cultural worlds around them" [125]. The advent of ever larger datasets and models makes it difficult to audit them for harms such as bias [9].

Machine Learning has long been built on centralized datasets, however, these datasets are not themselves ready to use, nor free from the need for curation and interpretation. Much direct hands-on work, often undervalued [128] and underpaid [120], is needed to make these datasets useful, underscoring both the interpretive nature of ostensibly objective AI, as well as the enormous resources needed to pay for this manual work [162]. Further, as data becomes increasingly important for building more accurate models, companies are becoming less open in sharing their datasets and corresponding metadata [116]. Centralization of the intellectual product of the world, often for the benefit of a few wealthy western corporations with access to sufficient compute, raises concerns of data colonialism [144].

2.4 Political Economy of Software

AI ethics scholarship increasingly scrutinizes the political economy of software. One frame for examining the political economy of software is through contemporary labor practices. Recent work surveys tech workers about ethical concerns they encounter in their work, and demonstrates that those concerns cannot simply be remedied with design changes— rather, workers lack power to challenge corporate incentives which lead to their concerns [163]. Others examine the role of increasing tech worker collective action in response to the absence of structural power [13, 109].

A growing body of literature examines the role powerful companies and their resources play in shaping digital possibilities. Srnicek [140] shows how modern tech companies act as "platforms" in order to orchestrate both other firms and their customers to their benefit. Dyer-Witheford et al. [49] critique AI from a Marxist perspective, examining how increasing labor automation entrenches capital. Luitse and Denkena [93] examine the ascendance of large language models, and demonstrates how they are "intertwined with the business model of big tech companies and further shift power relations in their favour." Widder et al. [162] examine and reject the claim that open source will "democratize" access to AI, given AI's dependence on concentrated resources— including compute, data, and labor— in the hands of corporate actors.

While companies often support open source projects that are important to corporate strategy [162], open source contributors are not always paid for their labor, creating the risk of "underproduction" of key digital infrastructures that depend on voluntary maintenance contributions [19, 20, 34]. Work increasing developer contributions in necessary infrastructure is an active area of research [91]. Elinor Ostrom proposes principles for the governance of common-pool resources [117], and we similarly theorize public source code as a material resource and commons [131]. Cooper et al. [32] demonstrates how in the nascent internet, the definition of "accountablity" expanded from acute monetary concerns to issues of wider policy for governing common resources. We similarly examine the inextricable link between historical developments, material resources and questions of governance.

3 THE HISTORY OF CENTRALIZATION IN OPEN-CODE

The historical provenance of datasets are a reoccurring concern in scholarship on fairness, accountability and transparency [57, 75, 92]. Nearly all recent models for generating code are trained on centralized open-code on GitHub [5, 56, 83, 87]. While scholars have examined these datasets on copyright grounds [39], we trace the ideological, legal, and technical origins of open source, attending in particular to how this movement led to centralized codebases. The communities that produced F/OSS code were often explicitly political, [30, 78] and scribal, assiduously documenting their introspective discourse. We use a variety of primary and secondary sources to build on prior work that traces the history of other political concepts such as accountability [33], fairness [72], and AI ethics [12].

3.1 The Ideological Origins of F/OSS

The Free Software Foundation (FSF), and the Open Source Initiative (OSI) shepherded new practices for hosting, distributing, and accessing publicly available source code [104]. The legal innovation of copyleft licensing, notably the FSF's General Public Licence (GPL) used "intellectual property rules to create a commons in cyberspace" [103], by enabling source code to be shared, used, and modified freely, with the "viral" condition that resulting software be distributed on the same terms. The FSF was founded as a moral crusade against proprietary software [141]. The OSI was founded in response, to reject moral crusades and instead emphasize the business benefits of openly sharing code [79]. They decided intentionally to "dump the moralizing and confrontational attitude [of] 'free software' [...] and sell the idea strictly on [...] pragmatic, business-case grounds" [145]. Though they had opposing ideological commitments [102], both worked— often collaboratively— on innovations that are technical (code), legal (licensing), and sociological (norms of sharing) [103, 104] to construct the F/OSS ecosystem.

3.2 Software Distribution and Package Management

Once written, software requires distribution channels to developers and users. While Microsoft can arrange with hardware manufacturers to have software pre-installed [47], this option was not available to the early F/OSS movement. The FSF's early years were funded by Richard Stallman selling and shipping tapes of code to users and contributors [104]. Mailing physical tapes was a common mode of distribution at the time [10].

Distributing software this way is inefficient, and impeded collaboration. ARPANET and later the Internet fundamentally changed this. Users could now post source and compiled code online. The Linux project began in 1991 to build a free operating system kernel when Linus Torvalds posted the "Sources for this pet project of mine [which] can be found at nic.funet.fi (128.214.6.100)" [105]. Collaborators began navigating to that address, downloading the code and making contributions they would send back to Linus. Linux's rapid adoption, popularity, and its organizational mythology helped to propel the growth of the broader F/OSS movements [123]. When Linux switched to the the FSF's GPLv2 license, Linus emphasized it was "it as an *engineering* choice and as a way to allow people to improve and share rather than as a moral imperative" [68]. Unlike other F/OSS projects such as GNU and BSD, which had a core team of co-located developers, Linux was maintained by distributed strangers online. Linux needed the GPL, and its licensing regime informed how it developed both technically and socially. As early as 1993, it was understood that Linux was not just a project but a methodology [104]. Working collaboratively, in public, became the basis of open source code as a "knowledge commons" [14, 131]. Originally, compressed source code was shared via web servers or mailing lists, [146, 167]. Around 1993, some distributions of Linux developed "packages" to allow people to install a pre-compiled binary using a package manager built into the operating system [104]. Debian released `apt-get` in 1998, making it possible to download a package and its dependencies at once [27]. This made it easier to develop software by compositing smaller modular components [82, 107]. In 2005, Linus released Git, a version control system designed to manage the unique challenges of the decentralized collaboration environment that Linux had pioneered.

Individual programming languages were facing similar issues around sharing code, libraries and packages. The LaTeX ecosystem was one of the first to centralize package management: beginning in 1993 developers could upload packages to the Comprehensive TeX Archive Network (CTAN) [64], a model adopted by other languages such R (CRAN) and Perl (CPAN) [148]. In the latter case, CPAN was considered the "killer app" for the language [130], allowing novices to build complex applications in few lines of code. Package management had thus become a necessary feature for new programming languages.

Package registries exhibit characteristics of a natural monopoly [124]. The cost of hosting packages is expensive at scale, and a single standard can aid in interoperability, further increasing the adoption of the registry through network effects. We can see how centralized package registries arise in the case of JavaScript. As a language and a product JavaScript was developed in a rush. The first prototype was made in 10 days [132]. In that rush, JavaScript was released without language tooling or package hosting. As JavaScript libraries began cropping up in the 2000's, they were typically hosted on public servers for anyone to download or use directly by including a script tag referencing a URL [126]. While a few free content delivery networks for JavaScript packages emerged in the early 2010s (CDNjs, jsDelivr and Google Hosted Libraries), the most significant development was the "node package manager" (`npm`), an open source project which incorporated as a start-up in 2014.

Node.js is a runtime environment for server-side JavaScript that became popular in the 2010s. `npm` was created to serve as a package registry for the ecosystem. Though Node and `npm` were intended for servers, web developers found its infrastructure useful for front-end development. In 2014 the `npm` blog noted that"32% of the packages in the top 50 [installed packages] (and 50% of the actual downloads) are front-end tools or frameworks" even though it was not an intended use case for the platform [153]. `npm` had become the de facto registry for JavaScript packages. Growing with the web, by 2017 it had become the largest package registry of any language [16], before being acquired by GitHub (and by extension, Microsoft) in 2020 [25]. The messy, distributed, and ad-hoc methods of software delivery that pre-dated `npm` was not a saleable asset. *It required a process of centralization to become one.*

3.3 Modern Social Coding

Open source ecosystems of peer production are increasingly centralized on package registries (i. e., `npm` [81] or PyPi [151]) and social coding platforms [168]. As the number of open source projects and communities expanded, this led to the development of infrastructures to organize communities, not just code. SourceForge, founded in 1999, was the first centralized home for F/OSS code, and let users upload software for free. By 2007, SourceForge hosted approximately 150,000 projects and was "the place to 'see and be seen' if you're an up and coming open-source project. It [has] developers chatting with developers, sharing, rubbing elbows, strutting their stuff, watching each other build. It's a global community of coder geeks" [95].

In 2008, in the same era YouTube, Facebook, and Twitter were becoming dominant, GitHub was founded as a "Social Coding" startup, explicitly layering community features on top of Git. Social features such as stars, follows, and public contribution trackers [98] incentivize coding as *content creation* [50]. These features increased both user productivity and burnout [122]. It rapidly supplanted SourceForge. By 2018, GitHub announced it hosted 100 million repositories [155]. In 2023 the platform surpassed 100 million registered users while articulating a vision to be "the home for all developers" [44]. Project maintainers pursue project-specific goals intrinsically linked to the broader ecosystem [43, 58] beyond writing code. Users transition into *contributors* over a process of socialization, often marked by rites of passage [48]. The sustainability of an open source project is reflected in its ability to endure, expand, and self-replicate [90] by guiding users through this integration process.

3.4 Centralization

GitHub's dominance in hosting open source software became unequivocal, and today's F/OSS ecosystem is extremely centralized. GitHub's nearest competitor, GitLab, has less than a third of the number of users, many of whom likely have GitHub accounts as well [44, 136]. GitHub is also a crucial tool in education [119], and hiring where developers often put their GitHub account on their CV and recruiters often request it directly [97].

Decentralization was core to the original F/OSS ethos [69]. John Gilmore, the Free Software advocate, programmer, and early GNU contributor famously stated "The net interprets censorship as damage and routes around it"[51]. GitHub, however, is not the internet, but a Limited Liability Company registered and operating in the US where it is subject to US laws. In 2019, developers living in countries under US Sanctions including Iran, Crimea, and Syria could not access their data, finding an alert notice stating that "Due to U.S. trade control law restrictions, your GitHub account has been limited" [89]. In 2020, GitHub complied with a DMCA notice from the Recording Industry Association of America to take down code used for downloading YouTube Videos [24]. After developer outcry GitHub reversed course and changed its copyright policies [62]. In these examples, GitHub assumes the role of an arbiter of what can and cannot exist on the largest platform for F/OSS code. While Git is a decentralized protocol [147], it has enabled code to be highly centralized on GitHub.

Programming language package management is similarly highly centralized. In 2016, open source developer Azer Koçulu was asked by messaging company *Kik* to remove his `npm` package sharing the same name [164]. After he refused, Kik's lawyers reached out to `npm`, who agreed to transfer the name to Kik. In response, Koçulu removed over 250 of his own open-source packages from `npm`. One such package, called *left-pad*, an unassuming 11 lines of code, was depended on by thousands of other packages. Koçulu was explicit in a blog post titled "I Just Liberated My Modules" [80]:

> "*This situation made me realize that NPM is someone's private land where corporate[sic] is more powerful than the people, and I do open-source because, Power To The People.*"

Removing *left-pad* wreaked havoc throughout the JavaScript ecosystem becoming known as the 11 lines that "broke the internet" [2]. `npm` took the "unprecedented" step of un-unpublishing his code [164] to restore the broader ecosystem. Similarly in 2022, a developer published an intentionally corrupted version of his own packages to protest corporations using open source software without contributing back [133]. Once again, `npm` reverted his changes, and GitHub suspended his account.

Once technical infrastructure is centralized, attempts to reintroduce decentralization are difficult. In 2018, Node.js creator Ryan Dahl identified ten design regrets he had about the language and ecosystem he had helped create [35]. Core among them was the way Node handled modules, the decision to include `npm` into Node, and the centralizing effect it had. Dahl demonstrated Deno, an alternative runtime to Node which obviated the kinds of supply chain problems and intermediaries that `npm` had introduced into the JavaScript ecosystem. Dependencies emulated script tags in the browser, being pulled in from a URL online directly rather than a centralized package manager intermediary. Deno's 1.0 release called attention to the fact that Node.js is "is fundamentally centralized through the NPM repository, which is not inline with the ideals of the web" [37]

Deno originally avoided `npm` compatibility on these grounds. However, over the course of 2022 [36] and 2023 [38], Deno would backtrack and support some cross-compatibility with `npm`— all while simultaneously reiterating how "JavaScript['s] reliance on a single centralized module registry conflicts with the web's decentralized nature" [38]. Integration with `npm` was a highly requested feature by developers that Deno could not ignore. The network effects of centralization are powerful: once developers are used to a standardized way of doing things, change is difficult.

4 ARTIFACTS OF CODE CENTRALIZATION

This work examines tools whose existence is predicated on the prior existence of large centralized code bases. Specifically, we adopt a *cross-case* analysis in accordance with Robert Gerring's methodology [59]. To that end, our approach is hypothesis *generating*, rather than hypothesis *testing*, emphasizing the causal mechanism whereby centralization of the software commons produces the tools examined. Cases were selected inductively and evaluated on whether or not they were typical of tools produced from centralized code. Where similar tools were substitutes, we selected the more popular one. We prioritized tools that operate within F/OSS ecosystems as infrastructure, though that was not a strict exclusion criteria. Although each case study is context-specific, they are selected to contribute to a theoretical generalization [170] about the changing landscape of open-code tools. In the following sections we provide relevant background on each case.

4.1 Crater, 2015

The Rust programming language aggressively commits to "stability without stagnation" across updates to the language and compiler [99, 149]. Crater was introduced in 2015 as a tool to help guarantee stability by "compiling and running tests for every crate on crates.io (and a few on GitHub)" [142]. Software packages— called crates in the Rust ecosystem are typically shared on `crates.io`— the default registry for downloading and sharing packages. Nearly every public repository on GitHub with a `Cargo.lock` file [15] is tested. Brian Anderson developed the initial version of Crater as "a tool to run experiments across parts of the Rust ecosystem." [142]. Crater runs at least once a week [15]. All logs are public and posted online. When proposed changes to the Rust compiler break existing projects, the team will revert or patch the changes. In 2019 AWS began sponsoring the EC2 compute costs for Crater. The maintainer for Crater [4] explained:

> "*We know it's not perfect. We don't test any kind of private code because of course we don't have access to your source code. But also we only test crates.io and GitHub, and not other repositories such as GitLab, mostly because nobody got around to write scrapers yet... [Crater is] the real reason why we can afford to make such fast releases... I wouldn't be comfortable making releases every six weeks without Crater because they would be so buggy.*"

4.2 Dependabot, 2017

To maintain dependencies and mitigate security vulnerabilities in the software supply chain, GitHub uses a system for automated pull requests called Dependabot [61]. Founded as a start-up in 2017, Dependabot was acquired by GitHub in 2019 and has been fully integrated into GitHub's platform [70]. Dependabot alerts are on by default for public repositories on GitHub [61, 100]. Security Advisories are synchronized from the National Vulnerability Database [108], and the GitHub Advisory Database. While other dependency management bots exist, Dependabot's integration on GitHub makes it the most widely used. In 2019, 67% of all bot-created pull requests came from a combination of the original and GitHub native versions of Dependabot [169].

4.3 Copilot, 2022

Copilot is a subscription cloud-based "artificial intelligence" assistant for writing code. It was originally based on a derivative of OpenAI's Codex model [21] trained on GitHub repositories. Paying subscribers can install Copilot as a plugin within several popular code editors, where it acts as a form of advanced auto-complete. The product is free to use for students and "verified open-source maintainers", where maintainer is defined as "someone who has write or admin access to one or more of the most popular open-source projects on GitHub" [44]. Based on GitHub's definition all maintainers use GitHub [74].

4.4 Case Study Justification

We investigate Crater, Dependabot, and Copilot based on their capacity to help clarify the *boundaries* of emerging open-code artifacts that are produced via centralization. From Gerring [59] we apply diversity sampling. Cases demonstrate diversity in their business model's and origins. Dependabot began as a startup that was acquired and integrated into GitHub, Crater is internal infrastructure for Rust's largely volunteer maintainers and Copilot is a subscription product. Whereas Dependabot and Crater originated with small teams, Copilot was predicated on significant capital, expertise, and data provided by Microsoft, OpenAI, and GitHub respectively.

The cases we select are also diverse in their interaction model. Individual programmers interact with Dependabot as as their code enters GitHub's platform. Copilot lives "with" a programmer in their IDE. In contrast, most Rust programmers will only notice that their code continues to compile, never necessarily attributing it to Crater.

Our case inclusion criteria required determining if a tool requires a large set of software generated via peer production as a functional input. We define those inputs as centralized open-code. We contrast this with popular F/OSS Libraries such as PyTorch, Serde, or zlib, which do not rely on centralized open-code to function. While they may be central nodes in dependency networks— both up and downstream of other projects— our inclusion criteria hinges on whether tools *rely* on the centralization of large datasets of open-code to achieve their stated utility. Dependabot, Crater, and Copilot all require the existence of extensive code ecosystems, producing large volumes of software. Centralized commons based tools must be in regular sync with the broader open source ecosystem. Crater, Dependabot, and Copilot are constantly re-run or retrained as new software is published online.

Each tool is also predicated on centralization in the F/OSS ecosystem. For example, as programmers increasingly depend on growing lists of open source dependencies [50, 139, 166], tools such as Dependabot become necessary for developers who could not otherwise keep up with updates to dependencies in real-time. In the case of the Rust ecosystem, Crater's maintainer states that they only use GitHub and Cargo, not taking the time to write scrapers for even the second largest public code forge. In a more distributed ecosystem, Crater would presumably look very different. For Copilot, the original model was trained on GitHub code [21], and has been extended via extensive retraining with augmentation from telemetry data [173]. While there are numerous datasets curated specifically for training code synthesis models, they are nearly all curated from public code on GitHub's Archive [5, 56, 83, 87].

Differences occur even in the tools produced by the same firm. Notably, Dependabot and Copilot are both released by GitHub. Dependabot is not the only bot intended to provide ecosystem-wide assistance. Other dependency management bots such as Greenkeeper (deprecated) and Snyk characterized by Wessel et al. [158] would make for promising case studies. While various languages employ tools to mitigate risks [110, 154], Dependabot, being integrated into Github and on by default makes it central infrastructure for the F/OSS ecosystem.

Alternative coding assistants such as Amazon's CodeWhisper, Tabnine, and ChatGPT [88] embody similar features as Copilot and could have been examined. Unlike ChatGPT, Copilot runs in user's existing development environment adding mode diversity to our cases. Copilot was selected for its documented scale [172] and its intentional integration with open source communities by granting free licenses to open source developers [74]. Additionally, its development by Microsoft and GitHub were considered, given Microsoft's role as arguably the primary beneficiary of centralized code on GitHub.

Across programming languages Crater is unique [135]. While centralized package mangers and tooling have become standard among many programming languages [11, 45, 148], Crater stands out by using its package management infrastructure, and public code as a resource for improving language development.

5 COMPARING ARTIFACTS OF CODE CENTRALIZATION

Each case is a product downstream of commons based peer production, centralization, aggregation, and synthesis loops. These artifacts are novel emergent phenomena that have not been categorized collectively. The variance across tools described in this section seeks to explore the bounds of this phenomenon. Specifically, we trace the idiosyncrasies of centralized commons based tools along axes of technology, community interaction, and legal compliance.

5.1 Stated Goals

Each case provides documentation, code and marketing materials that explicitly describe how the tool is intended to function. Crater was made to support the Rust language's commitment to stability [149]. Crater relies on GitHub and `crates.io`, to engage in "ecosystem-level experimentation" [6], treating the entirety of the open source Rust codebase as a test suite to evaluate proposed changes.

Dependabot primarily focuses on safety, aiming to strengthen the overall security of open source projects by ensuring that individual packages are updated and vulnerabilities are patched [106]. This stated goal extends to both upstream and downstream projects — a collective approach to security enhancement. In 2020, a blog post from Dependabot stated:

> "*Keeping dependencies updated is a crucial part of securing your software supply chain, whether you're working on an open-source project or a large enterprise. To make that easy, we're sticking to our promise to make all Dependabot features free for every repository on GitHub.*" [106]

Along these lines, Dependabot emphasizes its commitment to enabling security measures for an entire software supply chain by ensuring its features are freely accessible to every repository on GitHub.

Copilot's marketing home page describes it as an "AI pair programmer" [73]. It is primarily concerned with end-user task completion by automating repetitive coding tasks to enable developers to concentrate on more substantial development tasks. The GitHub blog's Copilot announcement underscores that "GitHub Copilot distills the collective knowledge of the world's developers into an editor extension that suggests code in real time, to help you stay focused on what matters most: building great software" [74].

Security and language stability for a broader ecosystem of users and tools are presented as integral goals for Dependabot and Crater. In comparison, Copilot's goals focus on efficiency and productivity for individual developers through artificial intelligence. The different ideological commitments of each tool demonstrate how centralization may be leveraged into either product-style tools or shared infrastructure.

5.2 Technology

While all three tools depend on the centralization of open source code, as described in Section 3.4, how they operationalize centralization differs. Here we compare how these tools leverage centralization to provide data as a resource, alongside their dependence on compute, and their core technical functionality, in Table 1.

The technical implementations of each tool varies in their transparency and reproducibility. Dependabot and Crater, for example, are themselves open source and both their code and underlying datasets are publicly available. Dependabot has been forked, reused, and integrated into other open source projects. Crater has inspired the creation of similar tooling [115] targeted at related concerns in the Rust ecosystem. Copilot has been emulated by companies [41] and open source projects [5, 86, 111, 112] such as FauxPilot [8, 26]

Crater depends on Rust's central package manager. Additional packages are sourced from GitHub. While the maintainers discuss the possibility of including other code hosting platforms, their choice not to do so suggests an implicit acceptance of GitHub's dominance [4]. Crater currently runs on an AWS c5.2xlarge with 2 TB of storage [142]. This costs $0.34/hour, and runs typically take a few days [15]. The cost is not insubstantial, "if the usage of Rust skyrockets [the Rust Foundation is] going to reach a point where it's not economically feasible to run Crater in a timely fashion" [142]. However, the resources are replicable by researchers [115] or other open source foundations.

Dependabot is a mechanism for interacting with the GitHub Advisory Database. Beyond the open source core logic for checking dependencies, Dependabot is largely a user interface to a database. GitHub scans user code and crosschecks it against their shared database [100]. In addition, GitHub is a numbering authority capable of adding entries to the Common Vulnerabilities and Exposures (CVE) system [96], the IT security industry standard venue for vulnerability disclosures.

Copilot is trained on code hosted on GitHub, without explicit consent from project owners and without regard for project licences. While others have used GitHub's Archive [5, 86, 111, 112] to train coding assistants, GitHub already hosts the code, making it technically trivial for them to access, providing an advantage given that dataset size yields significant advantages for transformer-like models [71]. Telemetry data from Copilot IDE plugin gives GitHub a substantial asset for improving and growing the model as developers' use of the tool becomes additional training data [84]. Training large models is financially costly, and while GitHub does not release details on training or compute costs, estimates based on the AWS EC2 P3 compute instances used by Allal et al. [5] suggest a lower bound of $40-$60,000, not including significant additional costs for multiple iterations during testing and inference while in use. Reportedly, Copilot looses money and depends on Microsoft's significant financial resources [46]. Copilot, like other large AI models are fundamentally products of large, well-resourced institutions.

5.3 Intellectual Property

In November 2022, a class-action lawsuit was filed against GitHub, Microsoft, and OpenAI alleging that Copilot was trained on copyrighted computer code without preserving required attribution and copyright notices [17, 22, 152]. The lawsuit, *Doe v. GitHub, Inc.* broadly gestures at how such violations will create future harms for copyright management [127]. The U.S. intellectual property law framework in this case serves as a useful reference for analyzing convergent and divergent legal perspectives on Crater, Dependabot, and Copilot. We examine license choices, lawful inputs and outputs in the context of U.S. software copyright, and potential violations of the Digital Millennium Copyright Act (DMCA) § 1202 [17]. § 1202 (a) of the DMCA restricts providing or distributing falsified copyright management information, and § 1202 (b) restricts the removal of copyright management information [17].

In software development, enhancing code functionality often involves adding features, optimizing performance, or fixing bugs. Expressive rights, in the context of code, refer to the freedom to express creative or innovative elements in the code, such as unique algorithms, creative solutions, or original design choices [66].

Crater is dually-licensed under Apache 2.0 and MIT licenses [143]. Programmers use and share their Rust code on `crates.io`. Crater conducts regression testing on the compiler [142], a practice that does not contravene the terms of use outlined in common F/OSS licenses. In other words, Crater enhances the code's functionality, but does not infringe upon its expressive rights. In doing so, Crater provides functional improvements to the code in a way that does not encroach upon the creative or expressive aspects of the original code. There is separation between the functional enhancements of Rust code, and the creative expression embedded in the code. From a fair-use perspective, this approach does not diminish the market value of the shared code, but amplifies its worth by guaranteeing its continued efficacy. Crater relies on open source packages for noncommercial purposes. Further, the transformative nature of Crater's operations (i.e. the output *builds* on copyrighted code in a different manner or for a different purpose from the ingested code) further underscores the absence of copyright concerns. Since Crater does not produce code outputs, they do not violate DMCA § 1202.

Dependabot is self-activating and on by default for public repositories. Dependabot's core source code and underlying database are available online [61] and released under the Prosperity Public License (PPL). The PPL creates strong limitations for commercial use by offering a restricted trial for such uses [100]. Dependabot does not create copyright infringement for code inputs, or risks for outputs, as its operational function is to maintain the sustainability of a given repository. Along these lines, Dependabot's technical processes would be classified as transformative use. Its use enhances the copyrighted code towards the goal of protecting packages from security risks and ensuring security over the ecosystem.

Copilot, in the context of its output, is a code recommendation system that *may* generate copyright-infringing works [22]. Along these lines, Copilot is not considered liable under U.S. copyright law until it is prompted to produce an output. This implies that the tool itself is not engaging in primary infringement; instead, it becomes relevant to the copyright liability when it is activated or prompted to generate some output by a user that may involve copyrighted material. As such, automated recommendation tools that copy code from the commons produce a significant source of risk to their users [17]. Further, Copilot is trained on code from repositories on GitHub which are released under numerous open source licenses. Butterick [17] claims that ingesting and distributing licensed materials without attribution, copyright notices, or licensing terms violates DMCA § 1202. Notably, GitHub admitted that the text of the GPL appears in Copilot's training data over 700,000 times [172], a license which specifically requires attribution.

Whereas licenses enable others to obtain rights to copy and use software, "Terms of Service" agreements are legal contracts that outline a service relationship between a customer and a corporation [85]. Copilot is a proprietary product developed by GitHub in collaboration with OpenAI, and is not released under an open source license. Instead, it is a commercial service provided by GitHub, and users are required to pay for access [129]. Ongoing litigation [17] will determine the legality of mining publicly available source code to power an AI-writing tool operating under a paid subscription model [22, 42].

5.4 Interface and Community Participation

Each tool operates at distinct levels of abstraction from developers, influencing how users interact with the tool, and the autonomy they have to opt-out. Crater, as a tool for maintainers, is experienced by Rust programmers indirectly through language features. While there are features for developers to request or audit Crater

Table 1: Technical Features: Dataset, Compute Resources, and Core Function

Tool	Dataset	Compute Resources	Core Function
Crater	Packages indexed on `crates.io`, and packages the scraper identifies as Rust code bases on GitHub; 561,346 repos as of 2023.	AWS c5.2xlarge with 2 TB of storage [142]. Their configuration would cost $0.34/hour on AWS with runs taking a few days.	Compiles each package under two versions of the compiler, runs its test suite and reports failures.
Dependabot	GitHub Advisory Database including CVEs, NVD, and language-specific alerts. Database is open source and open to contributions.	Difficult to estimate at scale. Dependabot-core can run in a docker environment without significant compute.	A library for automated pull requests based on changes to the database.
Copilot	159 GB of Python code from 54M public GitHub repositories for OpenAI's original Codex model. There have been multiple updates to Copilot, expanding its use beyond python to other languages, and likely including data collected from telemetry from over 1M users.	"GPT-3-12B relied on hundreds of petaflop/sdays of compute. Fine-tuning Codex-12B consumed a similar amount of compute" [21]. Trained on a custom cluster with 10,000 V100 GPU's. Estimated cost is $5M in compute. Additional resources costs accrue at inference.	A decoder only transformer model which probabilistically estimates the most likely next token in a sequence. The model is fine-tuned on code datasets.

runs, the tool is effectively invisible to the average Rust developer. Dependabot takes the form of notifications that developers must respond to within GitHub. Users can configure notification activity for Dependabot, allowing for a degree of control over their interaction. Copilot operates at an inter-textual [103] level, integrated into the developer's coding environment as they write, offering real-time code suggestions and providing a chat interface.

With respect to community participation, each tool offers very different levels of interaction in data creation, tool development, and tool use. Dependabot enables users to respond to automated pull requests, akin to contributing code, with configurable notification settings. The community can actively contribute to the security landscape by raising security alerts, a process open to any GitHub user [61]. GitHub explicitly incentivizes this mode of participation via social signaling, stating "contributors will get public credit on their GitHub profile once their contribution is merged!" [18]. When users raise a security alert, GitHub provides an option to request a CVE entry be opened on their behalf. In this act, GitHub lends its authority as a CVE partner to its users, making the alert available to the wider security community beyond GitHub.

Dependabot is enabled by default on public repositories but has documentation for opting out. Crater, however, lacks an explicit method to opting out. While not documented nor routine, developers could theoretically obfuscate their repositories by excluding a `Cargo.lock` file, or by opening a pull request on Crater and adding themselves to the list of blacklisted repositories. All Craters runs are public, allowing anyone to observe experiments, watch running tests, and see which packages failed to compile. Copilot is available to paying subscribers or free for verified students, teachers, and maintainers of popular open source projects hosted on GitHub. Utilizing Copilot necessitates enabling telemetry data collection, indicating a level of data participation required for tool access. The terms and conditions [60] for the Copilot technical preview state:

> "*When you edit files with the GitHub Copilot plugin enabled, file content snippets and suggestion results will be shared with GitHub and OpenAI and used for diagnostic purposes and to improve suggestions*".

As such, centralized commons based tools can have different governance structures in how they interface with their community, and the extent to which members can opt out from sharing their data with each tool.

5.5 Political Economy

According to Ostrom [117], governance models for common-pool resources rely on well-defined boundaries, tailored rules, monitoring, and collective decision-making mechanisms. However, the growth of centralized commons based tools challenge these conventional norms, demanding a reevaluation of governance in open source ecosystems.

Tensions related to motivation and autonomy within communities may arise. In the context of Crater and Dependabot, for example, the absence of clear rule-making processes and monitoring mechanisms can hinder autonomy and motivation, presenting obstacles to sustained participation.

Changes and developments in F/OSS ecosystems are not predicated on technological determinism. Rather, they are dependent on choices made by system developers. The way developers perceive the role of the open source ecosystem— as a collaborator, a commons, or a resource to be mined— will reflect on whether they choose adhere to or challenge the norms and principles of the ideologies embedded in open source development.

Dependabot's embedded politics are shaped by the project's stated commitment to enhancing security in open source projects. Even those not hosted on GitHub. Dependabot's README [1] has a section titled "Why Is This Public" where they state:

> "*If we were paranoid about someone stealing our business then we'd be keeping it under lock and key. Dependabot-Core is public because we're more interested in it having an impact than we are in making a buck from it.*"

In their commitment to making the tool available and compatible with their competitors, Dependabot prioritizes project impact over profit. Dependabot relies on GitHub's Advisory Database, a public centralized resource to address security vulnerabilities. Together, these choices reflect a prioritization of the collective security of the open source community.

Copilot is a predicated on well-funded institutional support. The development of Copilot was a collaborative effort involving Microsoft owned GitHub and OpenAI, which Microsoft has a significant financial stake in. The substantial resources the three companies have invested in Copilot's creation include the datasets and community goodwill obtained through GitHub, expansive compute infrastructure from Microsoft [93], and expertise from OpenAI [21]. The technical infrastructure of Copilot is intricately tied to the economic and strategic interests of its institutional backers [159]. This financial support not only influences the tool's technical capacity, but also underscores broader questions regarding the governance, ethics, and socio-political consequences associated with AI-driven technologies developed within the realm of well-funded institutions.

6 DISCUSSION

Our work traces the process of *centralization* within the F/OSS ecosystem. Through our case study analysis, we demonstrate how centralization can create the conditions in which novel modes of commons based tools emerge. Centralization is a *necessary* precursor to the development of such tooling. In the following section, we discuss the implications of our findings and directions for future work.

(1) Centralized commons based resources are too valuable to ignore. We expect similar artifacts to those discussed to proliferate. Within the last year alone we observe *Crater-Semver*, for example, which takes specific inspiration from Crater [115]. In another example, *Fleet Context* attempts to "embed the entire python ecosystem" [171]. So long as there is not a significant shift in the centralized state of platforms, we expect to see more examples that operate at a level of abstraction that encompasses entire ecosystems. Scale creates new opportunities and harms. Classifying and cataloging these emergent artifacts can aid in tracing the implications of centralization for computational systems broadly.

(2) Centralized commons based tooling can vary in reciprocity. Some tools are other regarding, and some are individual serving. Dependabot and Crater are specifically in *service* of the commons. Just as likely are artifacts that *expropriate* the commons. It is pertinent that we center the sustainability of the ecosystems from which these artifacts are derived. As the internet is rapidly filling with the outputs of generative models, the hazard of "model collapse" — Models trained on their own outputs [134] threatens the category of centralized commons based tools. Along these lines, centralization and its attendant tooling can both reify and contravene the recursivity [78] of F/OSS ecosystems. Through centralization, the commons can produce extreme value and novel tooling, but can, in the process, become vulnerable to appropriation and control.

(3) Package management, software sharing, and open collaboration platforms exhibit features of a natural monopoly [118]. We show how network effects at scale produce a new kind of value. The existence of centralized commons based tools strengths the incentive to centralize communities. Once an ecosystem has been centralized, it tends to remain so. Commons based tools are demonstrably useful, and it is financially valuable to be the platform where this centralization occurs. Recently valued at $4.5 billion dollars [101], HuggingFace has become the de facto home for open source models and the collaborative machine learning community [77]. They have achieved success by emulating the approach taken by GitHub a decade earlier, centering many of the social aspects of peer production. Future work should continue to trace the social, political, and economic consequences of monopolistic tendencies within open source ecosystems and how they shape technological development.

(4) To understand the impact of computational systems on society, it is pertinent to center the real world choices of developers. These decisions are not only influenced by technical considerations but also by socio-economic factors, ethical considerations, and evolving cultural norms of sharing [23, 30]. By focusing on the choices developers make in adopting and shaping technological tools, we gain insight into the broader landscape within which computational systems operate. Transformations in the working practices of developers and the technical tools they choose enables a deeper understanding of how these developments may intersect with societal values, power dynamics, and ethical considerations. This further informs scholarship on the political economy of the open source ecosystem.

(5) In this work we documented the historical processes that shaped centralization within the F/OSS commons and the software artifacts that centralization can produce. We note, however, that this process is occurring across the internet in a wide variety of non-software contexts. Many AI systems have been enabled by the centralizing [114] tendencies of large tech companies. These companies wield immense control over data and resources, and continue to shape the trajectory of a wide range of computational systems.

7 CONCLUSION

In this paper, we classify tools derived from the labor of open source communities as part of a broader analytic category. Copilot can be more coherently interpreted when situated adjacent to tools reliant on data from the centralization of open source ecosystems. Our research traces the historical and ideological roots of open-code centralization. Our comparative analysis of Crater, Dependabot, and Copilot hones in on the implications of these artifacts for infrastructural dependencies, community adoption and intellectual property. Such technical artifacts that can *only* be built via centralization. We demonstrate how centralized commons based tools can simultaneously strengthen the open source ecosystem and push the commons towards further centralization.

ACKNOWLEDGMENTS

We would like to thank our brilliant colleagues A.F. Cooper, David Goedicke, Benjamin Mako Hill, James Grimmelmann, Wendy Ju, Amrit Kwatra, Eben Moglen, Helen Nissenbaum, and Luis Villa for their insightful and constructive feedback throughout this project.

REFERENCES

[1] 2022. Dependabot README.MD. https://github.com/dependabot/dependabot-core (see page: 8)

[2] Rabe Abdalkareem. 2017. Reasons and drawbacks of using trivial npm packages: the developers' perspective. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 1062–1064. https://doi.org/10.1145/3106237.3121278 (see page: 4)

[3] Darren Abramson and Ali Emami. 2022. Interpreting docstrings without using common sense. (2022). https://www.fsf.org/licensing/copilot/interpreting-docstrings-without-using-common-sense (see page: 1)

[4] Pietro Albinim. 2019. Shipping a compiler every six weeks. https://www.pietroalbini.org/blog/shipping-a-compiler-every-six-weeks/ (see pages: 5, 6)

[5] Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. 2023. SantaCoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988* (2023). (see pages: 3, 6, and 7)

[6] Brian Anderson. 2017. How Rust is Tested. https://brson.github.io/2017/07/10/how-rust-is-tested (see page: 6)

[7] Hao Bai. 2022. A practical three-phase approach to fully automated programming using system decomposition and coding copilots. In *Proceedings of the 2022 5th International Conference on Machine Learning and Machine Intelligence*. 183–189. (see page: 1)

[8] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 500–506. (see page: 6)

[9] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Virtual Event Canada, 610–623. https://doi.org/10.1145/3442188.3445922 (see pages: 1, 2)

[10] Thomas J Bergin. 2006. The origins of word processing software for personal computers: 1976-1985. *IEEE Annals of the History of Computing* 28, 4 (2006), 32–47. (see page: 3)

[11] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. 2012. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145* (2012). (see page: 6)

[12] Abeba Birhane, Elayne Ruane, Thomas Laurent, Matthew S. Brown, Johnathan Flowers, Anthony Ventresque, and Christopher L. Dancy. 2022. The forgotten margins of AI ethics. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 948–958. (see page: 3)

[13] William Boag, Harini Suresh, Bianca Lepe, and Catherine D'Ignazio. 2022. Tech Worker Organizing for Power and Accountability. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 452–463. https://doi.org/10.1145/3531146.3533111 (see page: 2)

[14] Valeriia Boldosova. 2015. Looking beyond traditional network relationships: Online Subcontracting Platform as an unconventional tool for connecting and benefiting actors in the network. (2015). (see page: 3)

[15] Mara Bos. 2022. Do we need a "Rust Standard"? https://blog.m-ou.se/rust-standard/ Additional reference: https://blog.m-ou.se/rust-standard/. (see pages: 5, 6)

[16] Paul Brown. 2017. State of the Union: npm - Linux.com. https://www.linux.com/news/state-union-npm/ (see page: 4)

[17] Matthew Butterick. 2022. This CoPilot is stupid and wants to kill me. (see page: 7)

[18] Kate Catlin. 2022. GitHub Advisory Database now open to community contributions. https://github.blog/2022-02-22-github-advisory-database-now-open-to-community-contributions/ Accessed Date. (see page: 8)

[19] Kaylea Champion. 2022. Sociotechnical Risk in Sustaining Digital Infrastructure. In *Companion Publication of the 2022 Conference on Computer Supported Cooperative Work and Social Computing* (Virtual Event, Taiwan) *(CSCW'22 Companion)*. Association for Computing Machinery, New York, NY, USA, 232–236. https://doi.org/10.1145/3500868.3561406 (see page: 3)

[20] Kaylea Champion and Benjamin Mako Hill. 2021. Underproduction: An Approach for Measuring Risk in Open Source Software. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 388–399. https://doi.org/10.1109/SANER50967.2021.00043 (see page: 3)

[21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021). (see pages: 5, 6, 8, and 9)

[22] Madiha Zahrah Choksi and David Goedicke. 2023. Whose Text Is It Anyway? Exploring BigCode, Intellectual Property, and Ethics. *arXiv preprint arXiv:2304.02839* (2023). (see page: 7)

[23] Madiha Zahrah Choksi and James Grimmelmann. 2024. How Licenses Learn. *Forthcoming, Lewis & Clark Law Review* 28, 2 (2024). (see pages: 1, 2, and 9)

[24] Catalin Cimpanu. 2024. RIAA blitz takes down 18 GitHub projects used for downloading YouTube videos. *ZDNET* (23 Oct 2024). https://www.zdnet.com/article/riaa-blitz-takes-down-18-github-projects-used-for-downloading-youtube-videos/ (see page: 4)

[25] Thomas Claburn. 2020. Microsoft's GitHub absorbs NPM into its code-hosting empire: JavaScript library vault used by 12 million devs now under Redmond's roof. https://www.theregister.com/2020/03/16/microsofts_github_npm/ (see page: 4)

[26] Thomas Claburn. 2022. *FauxPilot: Like GitHub Copilot without Microsoft telemetry*. https://www.theregister.com/2022/08/06/fauxpilot_github_copilot/ (see page: 6)

[27] Maelick Claes, Tom Mens, Roberto Di Cosmo, and Jérôme Vouillon. 2015. A historical analysis of Debian package incompatibilities. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 212–223. (see page: 3)

[28] Jennifer Cobbe, Michael Veale, and Jatinder Singh. 2023. Understanding Accountability in Algorithmic Supply Chains. In *2023 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Chicago IL USA, 1186–1197. https://doi.org/10.1145/3593013.3594073 (see page: 2)

[29] E Gabriella Coleman. 2005. Three ethical moments in Debian. *Available at SSRN 805287* (2005). (see page: 2)

[30] E. Gabriella Coleman. 2013. *Coding Freedom: The Ethics and Aesthetics of Hacking*. Princeton University Press, Princeton. (see pages: 2, 3, and 9)

[31] Danish Contractor, Daniel McDuff, Julia Katherine Haines, Jenny Lee, Christopher Hines, Brent Hecht, Nicholas Vincent, and Hanlin Li. 2022. Behavioral Use Licensing for Responsible AI. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 778–788. https://doi.org/10.1145/3531146.3533143 (see page: 2)

[32] A. Feder Cooper, Emanuel Moss, Benjamin Laufer, and Helen Nissenbaum. 2022. Accountability in an Algorithmic Society: Relationality, Responsibility, and Robustness in Machine Learning. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Seoul Republic of Korea, 864–876. https://doi.org/10.1145/3531146.3533150 (see pages: 2, 3)

[33] A. Feder Cooper and Gili Vidan. 2022. Making the Unaccountable Internet: The Changing Meaning of Accounting in the Early ARPANET. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 726–742. https://doi.org/10.1145/3531146.3533137 (see pages: 2, 3)

[34] Daniel Curto-Millet and Alberto Corsín Jiménez. 2022. The sustainability of open source commons. *European Journal of Information Systems* (2022), 1–19. (see page: 3)

[35] Ryan Dahl. 2018. 10 Things I Regret About Node.js. https://www.youtube.com/watch?v=m3bm9tb-8ya (see page: 4)

[36] Ryan Dahl. 2023. Why We Added package.json Support to Deno. https://deno.com/blog/package-json-support (see page: 5)

[37] Ryan Dahl, Bert Belder, and Bartek Iwańczuk. 2020. *Deno 1.0*. https://deno.com/blog/v1 (see page: 5)

[38] Ryan Dahl and Alon Bonder. 2022. Big Changes Ahead for Deno. https://deno.com/blog/changes (see page: 5)

[39] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734. (see page: 3)

[40] Alexandre Decan, Tom Mens, and Philippe Grosjean. 2019. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering* 24, 1 (2019), 381–416. (see page: 1)

[41] Ankur Desai and Atul Deo. 2022. https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/ (see page: 6)

[42] Drew DeVault. 2022. GitHub Copilot and open source laundering. https://drewdevault.com/2022/06/23/Copilot-GPL-washing.html (see page: 7)

[43] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. 2021. What makes a great maintainer of open source projects?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 982–994. (see page: 4)

[44] Thomas Dohmke. 2023. 100 million developers and counting. https://github.blog/2023-01-25-100-million-developers-and-counting/ (see pages: 4, 5)

[45] Alan AA Donovan and Brian W Kernighan. 2015. *The Go programming language*. Addison-Wesley Professional. (see page: 6)

[46] Tom Dotan and Deepa Seetharaman. 2023. Big Tech Struggles to Turn AI Hype Into Profits. *WSJ* (Oct 2023). https://www.wsj.com/tech/ai/ais-costly-buildup-could-make-early-products-a-hard-sell-bdd29b9f (see page: 7)

[47] Jay Dratler Jr. 1995. Microsoft as an Antitrust Target: IBM in Software. *Sw. UL REv.* 25 (1995), 671. (see page: 3)

[48] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14 (2005), 323–368. (see page: 4)

[49] Nick Dyer-Witheford, Atle Mikkola Kjøsen, and James Steinhoff. 2019. *Inhuman Power: Artificial Intelligence and the Future of Capitalism.* Pluto Press. https://doi.org/10.2307/j.ctvj4sxc6 jstor:10.2307/j.ctvj4sxc6 (see page: 3)

[50] Nadia Eghbal. 2020. *Working in public: the making and maintenance of open source software.* Stripe Press San Francisco. (see pages: 1, 4, and 6)

[51] Philip Elmer-Dewitt and D Jackson. 1993. First nation in cyberspace. *Time* 6 (1993), 62–64. (see page: 4)

[52] Ivo Emanuilov. 2022. Artificial Intelligence and Intellectual Property Law. In *IMEC Workshop, Date: 2022/06/17-2022/06/17, Location: Antwerp.* (see page: 1)

[53] Nat Friedman. 2022. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/ (see page: 2)

[54] Brett M Frischmann. 2012. Intellectual Infrastructure. *B. Frischmann, Infrastructure: The Social Value of Shared Resources* (2012), 253. (see page: 2)

[55] Ben Gansky and Sean McDonald. 2022. CounterFAccTual: How FAccT Undermines Its Organizing Principles. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 1982–1992. https://doi.org/10.1145/3531146.3533241 (see page: 2)

[56] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027* (2020). (see pages: 3, 6)

[57] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92. (see page: 3)

[58] R Stuart Geiger, Dorothy Howard, and Lilly Irani. 2021. The labor of maintaining and scaling free and open-source software projects. *Proceedings of the ACM on human-computer interaction* 5, CSCW1 (2021), 1–28. (see page: 4)

[59] John Gerring. 2006. *Case study research: Principles and practices.* Cambridge university press. (see pages: 2, 5)

[60] GitHub. 2021. Telemetry terms - GitHub Docs. https://web.archive.org/web/20210704072124/https://docs.github.com/en/github/copilot/telemetry-terms (see page: 8)

[61] Github. 2023. Code Security: Dependabot. https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts Additional reference: https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts. (see pages: 5, 7, and 8)

[62] GitHub. 2024. Standing up for developers: youtube-dl is back. https://github.blog/2020-11-16-standing-up-for-developers-youtube-dl-is-back/. (see page: 4)

[63] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. 2022. On the rise and fall of CI services in GitHub. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 662–672. (see page: 1)

[64] G Grätzer. 2009. What Is New in LATEX? II. TEX implementations, Evolution or Revolution. *Notices of the AMS* 56, 5 (2009). (see page: 3)

[65] James Grimmelmann. 2009. The Internet is a semicommons. *Fordham L. Rev.* 78 (2009), 2799. (see pages: 1, 2)

[66] James Grimmelmann. 2015. Copyright for literate robots. *Iowa L. Rev.* 101 (2015), 657. (see pages: 2, 7)

[67] F S Grodzinsky, K Miller, and M J Wolf. 2003. Ethical Issues in Open Source Software. *Journal of Information, Communication and Ethics in Society* 1, 4 (nov 2003), 193–205. https://doi.org/10.1108/14779960380000235 (see page: 2)

[68] The VAR guy. 2016. Torvalds Talks about Early Linux History, GPL License and Money. https://web.archive.org/web/20170324170531/http://thevarguy.com/open-source-application-software-companies/torvalds-talks-about-early-linux-history-gpl-license-and- (see page: 3)

[69] Reinhard Anton Handler. 2018. Protocols of Control: Collaboration in Free and Open Source Software. *Technologies of Labour and the Politics of Contradiction* (2018), 175–192. (see page: 4)

[70] Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. 2022. Automating Dependency Updates in Practice: An Exploratory Study on GitHub Dependabot. *arXiv preprint arXiv:2206.07230* (2022). (see page: 5)

[71] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022). (see page: 7)

[72] Ben Hutchinson and Margaret Mitchell. 2019. 50 years of test (un) fairness: Lessons for machine learning. In *Proceedings of the conference on fairness, accountability, and transparency.* 49–58. (see page: 3)

[73] Github Inc. 2021. Introducing GitHub Copilot: your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/ (see page: 6)

[74] Github Inc. 2022. GitHub Copilot is generally available to all developers. https://github.blog/2022-06-21-github-copilot-is-generally-available-to-all-developers/ (see pages: 5, 6)

[75] Yacine Jernite, Huu Nguyen, Stella Biderman, Anna Rogers, Maraim Masoud, Valentin Danchev, Samson Tan, Alexandra Sasha Luccioni, Nishant Subramani, Isaac Johnson, Gerard Dupont, Jesse Dodge, Kyle Lo, Zeerak Talat, Dragomir Radev, Aaron Gokaslan, Somaieh Nikpoor, Peter Henderson, Rishi Bommasani, and Margaret Mitchell. 2022. Data Governance in the Age of Large-Scale Data-Driven Language Technology. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 2206–2222. https://doi.org/10.1145/3531146.3534637 (see page: 3)

[76] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J Cai, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems.* 1–19. (see page: 1)

[77] Wenxin Jiang, Nicholas Synovic, Matt Hyatt, Taylor R Schorlemmer, Rohan Sethi, Yung-Hsiang Lu, George K Thiruvathukal, and James C Davis. 2023. An empirical study of pre-trained model reuse in the hugging face deep learning model registry. *arXiv preprint arXiv:2303.02552* (2023). (see page: 9)

[78] Christopher Kelty. 2005. Geeks, social imaginaries, and recursive publics. *Cultural Anthropology* 20, 2 (2005), 185–214. (see pages: 3, 9)

[79] Mathias Klang. 2005. Free software and open source: The freedom debate and its consequences. *First Monday* (2005). (see page: 3)

[80] Azer Koçulu. 2016. I've Just Liberated My Modules. web.archive.org/web/20180217094442/http://azer.bike/journal/i-ve-just-liberated-my-modules (see page: 4)

[81] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, and Bogdan Vasilescu. 2020. Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 505–517. (see page: 4)

[82] Butler W Lampson. 2004. Software components: Only the giants survive. In *Computer Systems: Theory, Technology, and Applications.* Springer, 137–145. (see page: 3)

[83] Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, et al. 2022. The bigscience roots corpus: A 1.6 tb composite multilingual dataset. *Advances in Neural Information Processing Systems* 35 (2022), 31809–31826. (see pages: 3, 6)

[84] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems* 35 (2022), 21314–21328. (see page: 7)

[85] Mark A Lemley. 2006. Terms of use. *Minn. L. Rev.* 91 (2006), 459. (see page: 7)

[86] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023). (see pages: 6, 7)

[87] Renee Li, Pavitthra Pandurangan, Hana Frluckaj, and Laura Dabbish. 2021. Code of conduct conversations in open source software projects on github. *Proceedings of the ACM on Human-computer Interaction* 5, CSCW1 (2021), 1–31. (see pages: 3, 6)

[88] Yichen Li, Yintong Huo, Zhihan Jiang, Renyi Zhong, Pinjia He, Yuxin Su, and Michael R. Lyu. 2023. Exploring the Effectiveness of LLMs in Automated Logging Generation: An Empirical Study. arXiv:2307.05950 [cs.SE] (see page: 6)

[89] Rita Liao and Manish Singh. 2019. GitHub confirms it has blocked developers in Iran, Syria and Crimea. https://techcrunch.com/2019/07/29/github-ban-sanctioned-countries/ (see page: 4)

[90] Johan Linåker, Efi Papatheocharous, and Thomas Olsson. 2022. How to characterize the health of an Open Source Software project? A snowball literature review of an emerging practice. In *Proceedings of the 18th International Symposium on Open Collaboration.* 1–12. (see page: 4)

[91] Johan Linåker and Per Runeson. 2022. Sustaining Open Data as a Digital Common – Design Principles for Common Pool Resources Applied to Open Data Ecosystems. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3555051.3555066 (see page: 3)

[92] Alexandra Sasha Luccioni, Frances Corry, Hamsini Sridharan, Mike Ananny, Jason Schultz, and Kate Crawford. 2022. A Framework for Deprecating Datasets: Standardizing Documentation, Identification, and Communication. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 199–212. https://doi.org/10.1145/3531146.3533086 (see page: 3)

[93] Dieuwertje Luitse and Wiebke Denkena. 2021. The great transformer: Examining the role of large language models in the political economy of AI. *Big Data & Society* 8, 2 (2021), 20539517211047734. (see pages: 2, 3, and 9)

[94] Bradley M. Kuhn. 2022. If Software is My Copilot, Who Programmed My Software? https://sfconservancy.org/blog/2022/feb/03/github-copilot-copyleft-gpl/ (see page: 1)

[95] James Maguire. 2007. The SourceForge Story - Datamation. web.archive.org/web/20110804024950/http://itmanagement.earthweb.com/cnews/article.php/3705731 (see page: 4)

[96] David E Mann and Steven M Christey. 1999. Towards a common enumeration of vulnerabilities. In *2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, Indiana*. (see page: 7)

[97] Jennifer Marlow and Laura Dabbish. 2013. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 conference on Computer supported cooperative work*. 145–156. (see page: 4)

[98] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*. 117–128. (see page: 4)

[99] Niko Matsakis. 2014. *Semantic versioning for the language*. RFC 1122. https://rust-lang.github.io/rfcs/1122-language-semver.html (see page: 5)

[100] Mike McDonald. 2021. Goodbye Dependabot Preview, hello Dependabot. https://github.blog/2021-04-29-goodbye-dependabot-preview-hello-dependabot/ Additional reference: https://github.blog/2021-04-29-goodbye-dependabot-preview-hello-dependabot/. (see pages: 5, 7)

[101] Rachel Metz. [n. d.]. AI Startup Hugging Face Valued at 4.5 Billion After Raising Funding From Google, Nvidia. *Bloomberg* ([n. d.]). https://www.bloomberg.com/news/articles/2023-08-24/ai-startup-hugging-face-valued-at-4-5-billion-after-fundraising (see page: 9)

[102] Keith W Miller, Jeffrey Voas, and Tom Costello. 2010. Free and open source software. *It Professional* 12, 6 (2010), 14–16. (see page: 3)

[103] Eben Moglen. 1999. Anarchism triumphant: Free software and the death of copyright. *First Monday* 4, 8 (Aug. 1999). https://doi.org/10.5210/fm.v4i8.684 (see pages: 3, 8)

[104] Glyn Moody. 2009. *Rebel code: Linux and the open source revolution*. Hachette UK. (see page: 3)

[105] J Moon and Lee Sproull. 2010. Essence of distributed work. *Online communication and collaboration: A reader* 125 (2010). (see page: 3)

[106] Alex Mullans. 2021. Keep all your packages up to date with Dependabot. https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/ (see page: 6)

[107] Ian Murdock. 2007. How package management changed everything. https://web.archive.org/web/20090223072201/http://ianmurdock.com/2007/07/21/how-package-management-changed-everything/ (see page: 3)

[108] national institute of Standards and Security. 2023. National Vulnerability Database. https://nvd.nist.gov/ (see page: 5)

[109] Nataliya Nedzhvetskaya and J. S. Tan. 2022. The Role of Workers in AI Ethics and Governance. In *The Oxford Handbook of AI Governance* (1 ed.), Justin B. Bullock, Yu-Che Chen, Johannes Himmelreich, Valerie M. Hudson, Anton Korinek, Matthew M. Young, and Baobao Zhang (Eds.). Oxford University Press, C68.S1–C68.N14. https://doi.org/10.1093/oxfordhb/9780197579329.013.68 (see page: 2)

[110] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. 2022. Sigstore: software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2353–2367. (see page: 6)

[111] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. CodeGen2: Lessons for Training LLMs on Programming and Natural Languages. *ICLR* (2023). (see pages: 6, 7)

[112] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. *ICLR* (2023). (see pages: 6, 7)

[113] Helen Nissenbaum. 1996. Accountability in a Computerized Society. *Science and Engineering Ethics* 2, 1 (mar 1996), 25–42. https://doi.org/10.1007/BF02639315 (see page: 2)

[114] Mark Nottingham. 2023. *Centralization, decentralization, and internet standards*. Technical Report. RFC 9518. IETF. ht tp://tools. ietf. org/rfc/rfc9518. txt. (see page: 9)

[115] Tomasz Nowak, Michał Staniewski, Mieszko Grodzicki, and Bartosz Smolarczyk. 2023. Accelerating package expansion in Rust through development of a semantic versioning tool. *arXiv preprint arXiv:2308.14623* (2023). (see pages: 6, 7, and 9)

[116] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs] (see page: 2)

[117] Elinor Ostrom. 2008. Tragedy of the commons. *The new palgrave dictionary of economics* 2 (2008), 1–4. (see pages: 1, 2, 3, and 8)

[118] Frank A Pasquale. 2018. Tech platforms and the knowledge problem. *American Affairs, Summer* (2018). (see page: 9)

[119] Yasset Perez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J Eglen, Daniel S Katz, et al. 2016. Ten simple rules for taking advantage of Git and GitHub. , e1004947 pages. (see page: 4)

[120] Billy Perrigo. 2023. OpenAI Used Kenyan Workers on Less Than $2 Per Hour: Exclusive | Time. https://time.com/6247678/openai-chatgpt-kenya-workers/. (see page: 2)

[121] Ben Puryear and Gina Sprint. 2022. Github copilot in the classroom: learning to code with AI assistance. *Journal of Computing Sciences in Colleges* 38, 1 (2022), 37–47. (see page: 1)

[122] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. 57–60. (see page: 4)

[123] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (1999), 23–49. (see page: 3)

[124] Gil B Reschenthaler and Fred Thompson. 1996. The information revolution and the new public management. *Journal of public administration research and theory* 6, 1 (1996), 125–143. (see page: 3)

[125] David Ribes and Steven J. Jackson. 2013. Data Bite Man: The Work of Sustaining a Long-Term Study. In *"Raw Data" Is an Oxymoron*, Lisa Gitelman (Ed.). The MIT Press, 0. https://doi.org/10.7551/mitpress/9302.003.0010 (see page: 2)

[126] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. 2010. An analysis of the dynamic behavior of JavaScript programs. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1–12. (see page: 4)

[127] Emma Roth. 2023. Microsoft, GitHub, and OpenAI ask court to throw out AI copyright lawsuit. https://www.theverge.com/2023/1/28/23575919/microsoft-openai-github-dismiss-copilot-ai-copyright-lawsuit (see page: 7)

[128] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone Wants to Do the Model Work, Not the Data Work": Data Cascades in High-Stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15. (see page: 2)

[129] Pamela Samuelson. 2023. Legal Challenges to Generative AI, Part I. 66, 7 (jun 2023), 20–23. https://doi.org/10.1145/3597151 (see pages: 1, 7)

[130] Randal L Schwartz, Tom Phoenix, et al. 2012. *Intermediate Perl: Beyond The Basics of Learning Perl*. " O'Reilly Media, Inc.". (see page: 3)

[131] Charles M Schweik and Robert C English. 2012. *Internet success: a study of open-source software commons*. MIT Press. (see page: 3)

[132] Charles Severance. 2012. Javascript: Designing a language in 10 days. *Computer* 45, 2 (2012), 7–8. (see page: 4)

[133] Ax Sharma. 2022. Dev corrupts NPM libs 'colors' and 'faker' breaking thousands of apps. *BleepingComputer* (Jan 2022). https://www.bleepingcomputer.com/news/security/dev-corrupts-npm-libs-colors-and-faker-breaking-thousands-of-apps/ (see page: 4)

[134] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. 2023. The Curse of Recursion: Training on Generated Data Makes Models Forget. *arXiv preprint arxiv:2305.17493* (2023). (see page: 9)

[135] Joseph Sible, David Svoboda, and Garret Wassermann. 2023. Will Rust Solve Software Security? (2023). (see page: 6)

[136] Sid Sijbrandij. 2020. Upcoming changes to CI/CD Minutes for free tier users on GitLab.com. about.gitlab.com/blog/2020/09/01/ci-minutes-update-free-users (see page: 4)

[137] Henry E Smith. 2004. Property and property rules. *NYUL rev.* 79 (2004), 1719. (see page: 1)

[138] Irene Solaiman. 2023. The gradient of generative AI release: Methods and considerations. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*. 111–122. (see page: 2)

[139] César Soto-Valero, Thomas Durieux, and Benoit Baudry. 2021. A longitudinal analysis of bloated java dependencies. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1021–1031. (see page: 6)

[140] Nick Srnicek. 2016. *Platform Capitalism*. John Wiley & Sons. (see pages: 1, 2)

[141] Richard Stallman. 2009. Viewpoint Why" open source" misses the point of free software. *Commun. ACM* 52, 6 (2009), 31–33. (see page: 3)

[142] Rust Team. 2015. Crater. https://rustc-dev-guide.rust-lang.org/tests/crater.html Additional reference: https://github.com/rust-lang/crater. (see pages: 5, 6, 7, and 8)

[143] Rust Team. 2015. Crater. https://rustc-dev-guide.rust-lang.org/licenses.html (see page: 7)

[144] Jim Thatcher, David O'Sullivan, and Dillon Mahmoudi. 2016. Data Colonialism through Accumulation by Dispossession: New Metaphors for Daily Data. *Environment and Planning D: Society and Space* 34, 6 (dec 2016), 990–1006. https://doi.org/10.1177/0263775816633195 (see page: 2)

[145] Michael Tiemann. 2006. History of the OSI. web.archive.org/web/20090116020539/https://opensource.org/history Accessed: 2023-01-21. (see page: 3)

[146] Linus Torvalds. 1991. LINUX–a free unix-386 kernel. (see page: 3)

[147] Linus Torvalds. 2007. Tech Talk: Linus Torvalds on git. https://www.youtube.com/watch?v=4xpnkhjaok8 (see page: 4)

[148] Sam Tregar. 2002. CPAN. In *Writing Perl Modules for CPAN*. Springer, 1–20. (see pages: 3, 6)

[149] Aaron Turon and Niko Madsakis. 2014. Stability as a Deliverable. https://blog.rust-lang.org/2014/10/30/Stability.html Additional reference: https://blog.rust-lang.org/2014/10/30/Stability.html. (see pages: 5, 6)

[150] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by

large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7. (see page: 1)

[151] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 644–655. (see page: 4)

[152] James Vincent. 2022. The lawsuit that could rewrite the rules of AI copyright. https://www.theverge.com/2022/11/8/23446821/microsoft-openai-github-copilot-class-action-lawsuit-ai-copyright-violation-training-data (see page: 7)

[153] Laurie Voss. 2014. npm Blog Archive: npm and front-end packaging. https://blog.npmjs.org/post/101775448305/npm-and-front-end-packaging. (see page: 4)

[154] Duc-Ly Vu, Ivan Pashchenko, Fabio Massacci, Henrik Plate, and Antonino Sabetta. 2020. Typosquatting and combosquatting attacks on the python ecosystem. In *2020 ieee european symposium on security and privacy workshops (euros&pw)*. IEEE, 509–514. (see page: 6)

[155] Jason Warner. 2019. Thank you for 100 million repositories. (see page: 4)

[156] Paul V Weinstein. 2018. Why Microsoft is willing to pay so much for GitHub. *Harvard Business Review* 6 (2018). (see page: 2)

[157] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 172–178. (see page: 1)

[158] Mairieli Wessel, Bruno Mendes De Souza, Igor Steinmacher, Igor S Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A Gerosa. 2018. The power of bots: Characterizing and understanding bots in oss projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–19. (see page: 6)

[159] Meredith Whittaker. 2021. The steep cost of capture. *Interactions* 28, 6 (2021), 50–55. (see page: 9)

[160] David Gray Widder and Dawn Nafus. 2023. Dislocated Accountabilities in the "AI Supply Chain": Modularity and Developers' Notions of Responsibility. *SAGE Big Data & Society* 10, 1 (jan 2023), 20539517231177620. https://doi.org/10.1177/20539517231177620 (see page: 2)

[161] David Gray Widder, Dawn Nafus, Laura Dabbish, and James Herbsleb. 2022. Limits and Possibilities for "Ethical AI" in Open Source: A Study of Deepfakes. In *2022 ACM Conference on Fairness, Accountability, and Transparency* (Seoul, Republic of Korea) *(FAccT '22)*. Association for Computing Machinery, New York, NY, USA, 2035–2046. https://doi.org/10.1145/3531146.3533779 (see page: 2)

[162] David Gray Widder, Sarah West, and Meredith Whittaker. 2023. Open (For Business): Big Tech, Concentrated Power, and the Political Economy of Open AI. https://doi.org/10.2139/ssrn.4543807 (see pages: 2, 3)

[163] David Gray Widder, Derrick Zhen, Laura Dabbish, and James Herbsleb. 2023. It's about Power: What Ethical Concerns Do Software Engineers Have, and What Do They (Feel They Can) Do about Them?. In *2023 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Chicago IL USA, 467–479. https://doi.org/10.1145/3593013.3594012 (see page: 2)

[164] Chris Williams. 2016. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript. *The Register* 172 (23 Mar 2016). (see page: 4)

[165] Langdon Winner. 2017. Do artifacts have politics? In *Computer Ethics*. Routledge, 177–192. (see page: 1)

[166] Niklaus Wirth. 1995. A plea for lean software. *Computer* 28, 2 (1995), 64–68. (see page: 6)

[167] Ming-Wei Wu and Ying-Dar Lin. 2001. Open Source software development: An overview. *Computer* 34, 6 (2001), 33–38. (see page: 3)

[168] Yu Wu, Jessica Kropczynski, Patrick C Shih, and John M Carroll. 2014. Exploring the ecosystem of software developers on GitHub and other platforms. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*. 265–268. (see page: 4)

[169] Marvin Wyrich, Raoul Ghit, Tobias Haller, and Christian Müller. 2021. Bots don't mind waiting, do they? Comparing the interaction with automatically and manually created pull requests. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 6–10. (see page: 5)

[170] Robert K Yin. 2011. *Applications of case study research*. sage. (see page: 5)

[171] Andrew Zhou and Nicolas Ouporov. 2023. Building a RAG Pipeline for the Entire Python Ecosystem. https://fleet.so/blog/library-rag. (see page: 9)

[172] Albert Ziegler. 2022. GitHub Copilot research recitation. https://github.blog/2021-06-30-github-copilot-research-recitation/ (see pages: 6, 7)

[173] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 21–29. (see page: 6)