The Value of Open Source Software: The Infrastructure of Artificial Intelligence

Manuel Hoffmann[a] Frank Nagle[b] Yanuo Zhou[c]

This version: February 2, 2026

Abstract

Assessing the value of non–pecuniary (free) products is difficult, particularly for open source software (OSS), a global public good that plays a vital role in the economy and is foundational for most technologies we use today, including artificial intelligence (AI). However, it is difficult to measure the value of OSS and therefore, OSS remains largely unaccounted for in economic measures. Although prior studies have estimated the supply-side costs to recreate this software, a lack of data has hampered estimating the much larger demand-side (usage) value. As such, to understand the economic and social value of widely used OSS, we leverage unique global data from two sources capturing OSS usage by millions of global firms. We first estimate the supply-side value by calculating the cost to recreate the most widely used OSS once. We then calculate the demand-side value based on a replacement value for each firm that uses the software and would need to build it internally if OSS did not exist. We estimate the supply-side value of widely used OSS is $4.15 billion, but the demand-side value is much larger at $8.8 trillion. We find that firms would need to spend 3.5 times more on software than they currently do if OSS did not exist. Examining the nuances of this value, we find that the top six programming languages in our sample comprise 84% of the demand-side value of OSS and that 96% of the demand-side value is created by only 5% of OSS developers. Further, we find that the value of OSS technologies that undergird the rapid growth of the AI economy had already reached $1.46 trillion prior to the release of ChatGPT. As such, this paper quantifies the value of the infrastructure of the digital economy and provides a unique method of understanding the value of the infrastructure that made the AI explosion possible.

JEL Classification: H4; O3; J0
Keywords: Open-source software, global public good

Acknowledgement: The authors are grateful for financial and administrative support from the Linux Foundation without which the data from the Census would not have otherwise been available. We greatly appreciate the support of the Research Computing Services at Harvard Business School, the Laboratory for Innovation Science at Harvard, the Linux Foundation, and the software composition analysis data providers Snyk, the Synopsys Cybersecurity Research Center, and FOSSA. We thank Tianli Li and Misha Bouzinier for excellent research assistance. We are also thankful to the software developer Boris Martinovic as well as Rich Lander and Scott Hanselman from Microsoft for insights into the .NET ecosystem. We received helpful feedback from participants at the Harvard Business School Values and Valuations Conference, the Harvard Business School D[3] Research Day, and the 2023 Academy of Management Conference.

[a] Paul Merage School of Business, University of California, Irvine, California 92697, E-Mail: manuel.hoffmann@uci.edu.
[b] Initiative on the Digital Economy, Massachusetts Institute of Technology, Cambridge, MA 02142, E-Mail: fnagle@mit.edu fnagle@mit.edu.
[c] Rotman School of Management, University of Toronto, 105 St. George Street, Toronto, ON, M5S 3E6, Canada, E-Mail: yanuo.zhou@rotman.utoronto.ca.

1. Introduction

In 2011, venture capitalist Mark Andreessen famously argued that "software is eating the world" (Andreessen, 2011), a sentiment few would argue with today. More recently, venture capitalist Joseph Jacks argued that "open source is eating software faster than software is eating the world," (Jacks, 2022). Other recent studies have come to similar conclusions showing that open source software (OSS) – software whose source code is publicly available for inspection, use, and modification and is often created in a decentralized manner and distributed for free – appears in 96% of codebases (Synopsys 2023), and that some commercial software consists of up to 99.9% freely available OSS (Musseau et al., 2022). Although in its early days OSS frequently copied features from existing proprietary software, OSS today includes cutting edge technology in various fields including artificial intelligence (AI), quantum computing, big data, and analytics. However, despite the increasing importance of OSS to all software (and therefore to the entire economy), measuring its impact has been elusive. Traditionally, to measure the value created by a good or service, economists multiply the price (p) times the quantity sold (q). However, in OSS, p is generally zero since the source code is publicly available, and q is unknown due to the limited number of restrictions around how the code may be copied and reused. For example, if a company downloads a piece of OSS from a public code repository, it may copy it thousands of times internally (legally) and then share it with suppliers or customers (also legally), so public download data is insufficient. Although some recent studies have sought to estimate the value of p (discussed below), data for estimating q has been unavailable or intractable for anything more than just a handful of OSS packages. Using newly collected data from multiple sources, the goal of this paper is to provide estimates for both p and q and to use those to shine light on the question: What is the value of open source software?

Understanding the value of OSS is of critical importance not only due to the role it plays in the economy, but also due to it being one of the most successful and impactful modern examples of the centuries old economic concept of "the commons" which run the risk of meeting the fate known as "the tragedy of the commons." This concept can trace its roots as far back as the 4th-century BC philosopher Aristotle who wrote "That which is common to the greatest number gets the least amount of care. Men pay most attention to what is their own and care less for what is common." (Aristotle, 1981). William Forster Lloyd (1833) resurfaced the idea in modern economic thought by highlighting the example of shared tracts of land used for cattle grazing by

multiple cattle herders who each had an incentive to overuse the shared resource. Garrett Hardin (1968) brought the concept into the broader zeitgeist when he wrote an article discussing the problem entitled "The Tragedy of the Commons." Building on this work, Elinor Ostrom won the Nobel Prize for her research highlighting paths to avoiding the tragedy of the commons through community coordination efforts that did not necessitate government enforced laws to manage and guard the commons (Ostrom 1990). The parallels between shared grazing lands and shared digital infrastructure are palpable – the availability of communal grass to feed cattle, and in turn feed people, was critical to the agrarian economy, and the ability to not have to recreate code that someone else has already written is critical to the modern economy. Further, in both contexts, despite knowing grass and code are critical inputs to the economy, measuring their actual value is difficult. And, as the renowned mathematician and physicist Lord Kelvin is believed to have said, "If you can't measure it, you can't improve it." And if you cannot improve and maintain it, such common goods may crumble under the weight of their own success as they are overused, but underinvested in (Lifshitz-Assaf and Nagle, 2021). Therefore, measuring the value OSS creates is crucial to the future health of the digital economy, and the rest of the economy that is built on top of it.

Importantly, recent studies have attempted to address these measurement issues but fall short of capturing both the breadth and depth of OSS usage – a gap we seek to fill with this paper. For example, researchers have attempted to gain breadth by using novel methodologies to estimate the labor replacement value of the current corpus of OSS created in the United States at $38 billion in 2019 (Robbins et al., 2021) and that created in the European Union at €1 billion (Blind et al., 2021) by imputing the labor costs that it would have taken to rewrite existing OSS. Such efforts do a very good job at estimating what it would cost to replace all existing OSS if it disappeared tomorrow. However, the resultant estimates rely on two important assumptions. First, that all OSS is equally valuable from a usage standpoint, and second that the concept of OSS would still exist, and society would just need to rewrite the code once, thus addressing the aforementioned problem of a missing value for p, but not addressing the missing value of q. In a world where OSS did not exist at all, then each piece of OSS software would not need to be rewritten just once, but instead would need to be rewritten by every firm that used the software (assuming the firm could freely share the software within its boundaries). Other research (Greenstein and Nagle, 2014; Murciano-Goroff, Zhuo, and Greenstein, 2021) has gone deeper into this hypothetical, albeit in a narrow

manner, by only focusing on web servers (which are public facing on the Internet and can therefore be readily measured). Using different methods, both studies measure q for this one type of software and impute p by using a goods replacement value approach based on the prices for closed-source alternatives offered by firms. With data from the United States the resulting estimates show a value of $2 billion for the OSS Apache Web Server in 2012 (Greenstein and Nagle, 2014) and a combined value of $4.5 billion for Apache and the increasingly popular OSS web server nginx in 2018 (Murciano-Goroff, et al., 2021). However, although web servers are an important part of the OSS ecosystem, they constitute a small portion of it. We seek to build upon the important contributions this existing research has made in an attempt to go both broad and deep to create a more complete measure of the value of OSS.

To consider the value of OSS in both a broad and deep manner, we use data from two primary sources that allow us to gain insights into the OSS used at tens of thousands of firms across the world. The first is the "Census II of Free and Open Source Software – Application Libraries" (Nagle et al., 2022). The Census II project utilized partnerships with multiple software composition analysis (SCA) firms to create various lists of the most widely used OSS. SCAs are hired to scan the codebases of a company to ensure they are not violating any OSS licenses and, as a byproduct, track all the OSS code used by their customers and the products they build. The Census II project aggregated data from multiple SCAs to build a dataset of OSS usage at tens of thousands of firms based on millions of data points (observations of OSS usage). The second data source is the BuiltWith dataset, from which we leverage scans of nearly nine million websites to identify the underlying technology deployed by these websites, including OSS libraries. The BuiltWith data has been used in multiple academic studies (DeStefano & Timmis 2023, Dushnitsky & Stroube 2021, Koning et al.. 2022), but to our knowledge this is the first one to focus on OSS usage. The Census II data and the BuiltWith data are complementary as the former focuses on OSS that is built into the software a company sells, while the latter focuses on OSS that is built into a company's website, thus reducing the chances of double-counting observations across the datasets. In aggregate, these two datasets combined create the most complete measurement of OSS usage (q) to date. Further, by focusing on OSS that is widely deployed and used by firms, rather than considering all the projects that exist in an OSS repository, we enhance the methodologies of prior studies by reducing the likelihood of measurement error stemming from projects that are posted as publicly available OSS but are not actually used in any practical manner. Not accounting for this

measurement error would lead to overestimation of the actual value of OSS as projects that are widely used would be valued in the same way as projects that are not used at all.

To estimate *p*, we follow the literature discussed above and use the labor replacement value. First, we calculate the labor cost it would take an individual firm to recreate a given OSS package by measuring the number of lines of code within the package and then applying the Constructive Cost Model II (Boehm, 1984; Boehm et al., 2009) – also known as COCOMO II – to estimate the number of person-hours it would take to write the code from scratch. We then utilize global wage data from Salary Expert to get an accurate estimate of the labor costs a firm would incur if this piece of OSS did not exist. These costs can be combined with the *q* values from above at the OSS package level to estimate the combined value of all OSS from both the supply and demand sides.

We find a value ranging from $1.22 billion to $6.22 billion if we were to decide as a society to recreate all widely used OSS on the supply side. However, considering the actual usage of OSS leads to a demand-side value that is orders of magnitude larger and ranges from $2.59 trillion to $13.18 trillion, if each firm who used an OSS package had to recreate it from scratch (e.g., the concept of OSS did not exist). We document substantial heterogeneity of the value of OSS by programming language and internal vs outward-facing programming efforts. Further, we find considerable heterogeneity in value contributions by programmers as 5% of programmers are responsible for more than 90% of the value created on the supply- and demand- side. We show that in 2020 (prior to the release of ChatGPT and the resultant AI boom) the open source software environment already created an infrastructure value of over $1.4 trillion upon which AI investments could build and allowed, in part, for the rapid growth of the AI economy. The data we use is arguably the most comprehensive source of data to measure the value created by firm usage of OSS at this time. However, as for any project, the evidence is not complete and we argue that we underestimate the value since our data, e.g., does not include operating systems, which are a substantial omitted category of OSS.

This study makes four important contributions to the academic literature, practitioners, and policy makers. First, it provides the most complete estimate of the value of widely used OSS to date by accounting for not only the supply-side of OSS (price to create it), but also for the usage/demand-side at a scale that has not been done before. While prior estimates of the value of OSS only went either broad (estimating the supply-side costs of a large swath of OSS) or deep

(estimating the value created by one particular type of OSS), this study does both by using unique datasets that allow for a better understanding of the breadth and depth of OSS usage. Further, rather than measuring the value of all OSS, this study focuses on the value of OSS that is used by firms to create its products and websites, limiting the measurement error occurring in studies that are unable to account for which OSS is actually used in production. This contribution builds upon, and extends, important research (e.g., Blind et al., 2021; Greenstein and Nagle, 2014; Murciano-Goroff et al., 2021; Robbins et al., 2021) that has sought to identify the value of this vital resource that contributes a great deal to the modern economy despite the difficulties measuring this contribution. In doing so, it adds insights to a long-running discussion related to the impact of information technology (IT) on productivity (Brynjolfsson, 1993; Brynjolfsson and Hitt, 1996; Nagle, 2019a; Solow, 1987) known as the "productivity paradox" where IT investments can have limited impact on productivity statistics. This debate has continued into the emerging context of AI (Brynjolfsson, Rock, and Syverson, 2018). Our work contributes to this conversation by highlighting a massive societal level cost-savings (and hence productivity enhancement) that is created by the existence of OSS. Furthermore, we introduce a novel methodology for assessing the economic potential of artificial intelligence value by classifying open source software packages into their AI relevance to measure their potential economic impact and by providing a framework to quantify the latent value of the foundational technologies that made the 2022 AI explosion possible.

Second, our research contributes methodological advances to the study of intangible capital by highlighting novel sources of data related to investments in OSS. Prior research has shown that intangible capital plays an increasingly important role in economic growth (Corrado, Hulten, and Sichel, 2009) and firm value (Peters and Taylor, 2017), but it often goes unmeasured or misattributed (Eisfeldt and Papanikolaou, 2014). Further, we demonstrate how these data sources can be used to understand the true investments in software that a firm makes, and that which they would have to make if OSS did not exist. This is valuable as investments in software are an increasingly important type of intangible capital that is driving innovation (Branstetter, Drev, and Kwon, 2019) and performance (Krishnan et al., 2000).

Third, our results help highlight for firms and managers the importance of OSS to their production, and ideally add weight to arguments that users of OSS should not just free ride but also contribute to the creation and maintenance of OSS (e.g., Henkel, 2008; Nagle, 2018). Such contributions are a fraction of the costs that firms would incur if OSS did not exist and the active

participation of OSS users in helping maintain the OSS they use is critical to the health and future well-being of the OSS ecosystem (Lifshitz-Assaf and Nagle, 2021; Zhang et al., 2019).

Fourth, and finally, our study helps inform policymakers who have recently started to understand the growing importance of OSS to the economy and taken actions to support the ecosystem (European Commission, 2020; Executive Order No. 14028, 2021). However, most of these efforts are related to securing the existing OSS ecosystem, which is quite important, but do not go as far as supporting the creation of new OSS. Our findings suggest that government support should also focus on the maintenance of the open source infrastructure that serve as a public good for all AI development. Our results help shine light on the importance of OSS to the overall economy and add weight to calls for more societal support of this critical resource. Our results further show that the majority of the value created by OSS is created by a small number of contributors. Although it has long been known that a small number of OSS contributors do most of the work, we add new insights that show this is even more true for the value creation of widely-used OSS projects and that societal support for these individuals is critical to the future success of OSS, and in turn, the economy.

The remainder of this paper is organized as follows. Section 2 describes the empirical setting and data. Section 3 discusses the methods including measurement challenges. In section 4, we estimate the value of open source software. Section 5 concludes.

2. Empirical Setting and Data

Although the concept of free and open software has existed since the 1950's, it became more popular in the 1980's due to the efforts of Richard Stallman and his launch of the GNU Project and the Free Software Foundation (Maracke, 2019). However, it was in the 1990's that OSS took off after Linus Torvalds released the Linux kernel, a now widely adopted OSS operating system (Tozzi, 2016). Today, OSS is considered a key building block of the digital economy and is widely used by software developers in everything from phones to cars to refrigerators to cutting-edge AI (Lifshitz-Assaf and Nagle, 2021).

We use two complementary main data sources to estimate the value of OSS. The first is the Census and is inward facing. It allows us to identify OSS code that goes into products that firms create. The second dataset is BuiltWith and is outward facing, allowing us to identify OSS code that consumers directly interact with through firm websites. Since the raw data in both datasets only contains package names, version numbers, and package manager names, and does

not contain any source code related information, we first obtain the publicly accessible code repository for each package which includes package-level information including the lines of code and the programming languages used. One may worry about double-counting for the value calculations as a result of using two separate datasets. However, the overlap of both the Census and the BuiltWith sample is very small: there are only 18 packages found in both datasets. Moreover, it is unlikely that double counting is a concern as the two datasets capture different dimensions of OSS usage: the Census captures packaged software whose usage is inward-facing while BuiltWith captures usage in websites that are outward-facing. Finally, as a supplementary dataset, we use GHTorrent, a detailed history of OSS-related activity on GitHub, the most popular OSS hosting platform and a commonly used data source for studies of OSS (e.g., Burton et al, 2017; Conti, Peukert, and Roche, 2023; Fackler, Hofmann, and Laurentsyeva, 2023; Kim, 2020; Tang, Wang, & Tong, 2023). This detailed historical data allows us to go deeper into how the value of OSS is created by better understanding the dispersion of the contributions across individual OSS developers. We describe the details of all three data sources and their preparation for estimation below.

Census. The Census II of Free and Open Source Software (here: Census) was jointly undertaken by the Linux Foundation and the Laboratory for Innovation Science at Harvard (Nagle et al., 2022).[1] The Census was created via the aggregation of data from three major software composition analysis (SCA) firms with thousands of clients across the globe. SCAs are hired to scan the codebase of a client and gather the OSS usage embedded in their proprietary software to ensure they are not violating any OSS license agreements.[2] Frequently this takes place as part of the due diligence process related to mergers and acquisitions. Unlike other OSS demand measures available from public sources such as package download counts and code changes, the way this data was collected ensures that we observe the precise amount and type of internal OSS usage of the firms. In addition, it allows us to trace the dependencies each package relies on, so that we can observe the indirect OSS usage that is commonly hidden and difficult to obtain.[3] The result is over

[1] The precise methodologies for the data collection and aggregation are detailed in the Census report by Nagle et al (2022).

[2] OSS usage licenses vary a great deal and while some licenses are very open and allow the code to be reused in any manner, including within proprietary code that will be made available for sale at a non-zero price, other licenses restrict reuse to only be allowed if the resultant code is released under the same OSS license (known as copyleft). A detailed discussion of OSS licenses can be found in Lerner and Tirole (2005) and Almeida et al (2017).

[3] Indirect OSS usage is captured by dependency analysis and is necessary to accurately measure the full breadth of the OSS a firm relies on. For example, if a firm's proprietary code calls OSS package A, but package A, in turn, calls

2.7 million observations of OSS packages being used within products created by the SCA client firms for the calendar year 2020 (January 1 to December 31, 2020).[4]

The Census project standardized package names based on the naming system of libraries.io – a widely used site maintained by Tidelift that organizes information about more than eight million open source packages. The Census focused on the top 2,000 packages based on usage reported from the three of the most prominent SCA vendors to identify the most widely used OSS rather than the long-tail of the usage distribution. This led to packages with less than five observations of usage being dropped. Since packages written in the JavaScript programming language, and usually hosted on the Node Package Manager (NPM), are generally smaller (fewer lines of code) than packages in other languages and therefore frequently have higher usage numbers (since developers must include many small packages instead of a few large packages), the Census separately selected the top 1,000 NPM packages and the top 1,000 non-NPM hosted packages.

This final dataset covers 70% of the total usage of OSS observed in the raw data of the census.[5] For each of these 2,000 OSS packages in the Census, we identified the raw code maintained on GitHub, the most widely used platform for hosting OSS.[6] We first attempted to obtain the GitHub repository uniform resource locator (URL) for each package from libraries.io. We were able to match 1,657 packages to repositories via this initial method.[7] For URLs without a matching repository, we performed Google searches loosely following the method in Singh (2020). More specifically, for each unmatched package, we used the Google API to search for the package name and "GitHub Repository" and treated the first GitHub repository URL in the results

package B, then only looking at the direct calls would miss that package B was a required building block for the firm's proprietary code.

[4] The Census defines an OSS package as "a unit of software that can be installed and managed by a package manager," and defines a package manager as "software that automates the process of installing and otherwise managing packages" (Nagle et al, 2022).

[5] The packages that made up the remaining 30% of the full Census data in the long-tail usage distribution were not shared in the final report and therefore cannot be included in our analysis.

[6] GitHub is a hosting and collaboration platform that contributors can use to coordinate the development and distribution of OSS projects. Founded in 2008, GitHub has become the largest hub for OSS development in the world. In January 2023, GitHub had more than 370 million repositories and over 100 million developers. In addition to personal users, a wide range of private firms actively use the GitHub platform, including Microsoft (which bought GitHub in 2018), Facebook, Google, and numerous other small and large firms.

[7] We attempted to access all repository URLs obtained as a sanity check to ensure they are in working condition. For those we could not access on GitHub, we manually found the correct URLs.

as its best matched GitHub URL.[8] This resulted in an additional 174 packages matched to a repository. Finally, we manually searched for URLs for the remaining 169 packages to identify the relevant repository. This entire process resulted in matching 1,840 out of the 2,000 Census packages to a code repository with the raw source code for the package. The unmatched packages were determined to have been either removed from GitHub or became proprietary (and thus the original source code was no longer available) and hence, the manual search allowed us to drop these 160 unmatched packages (less than 8% of the Census sample of 2,000) with high confidence from our analysis.

BuiltWith. The BuiltWith data contains scans of all public websites across the globe and identifies the technologies they use. Unlike the inward facing Census data which focus on the usage of OSS, the BuiltWith data scan for the use of both proprietary and OSS in firm websites without explicit differentiation. To separate OSS from proprietary software in BuiltWith, we turn to the subset of all open source web development software in the technology category that includes "JavaScript and its libraries," which generates 778 observations corresponding to the NPM OSS category in the Census data. There are two reasons for this sampling choice. First, this given category is constructed by BuiltWith and we use it as a proxy for OSS to separate it from pecuniary software. Second, JavaScript, one of the core technologies for building websites, is the most popular programming language by usage on GitHub (GitHub, 2022) and thus enables us to capture the most important OSS from the demand side perspective. The scans include 8.8 million unique websites and 72.8 million corresponding observations of OSS usage from January 1 to November 16, 2020. Further, to ensure that we measure the value of OSS usage generated by the private sector, we match the adopting domains of the JavaScript-related OSS from BuiltWith with company websites recorded in Orbis, Compustat, and PitchBook, three commonly used databases of corporate activity that capture registered businesses across the world. Performing this match ensures that OSS used by non-commercial websites (e.g., an individual person's personal website) is excluded in our analysis. This results in a match-rate of 38.6%, which corresponds to around 3.4 million websites of distinct firms.

[8] As a robustness check, we randomly selected 50 package-repository matches derived from the Google Search method and we checked them by hand. All matches were manually confirmed to be correct providing additional support for the automated matching method.

For the BuiltWith data, we cannot employ the first method we use for the Census (using libraries.io to help identify the repository URL) since only the technology names associated with the packages were provided by BuiltWith, and other information (e.g., package and package manager names) is not included. [9] Hence, we start by performing the Google Search method mentioned above which results in a match of 695 packages to repositories. We then manually searched the Github URLs for 83 of the remaining unmatched packages, resulting in an addition of 46 packages. In total, for the BuiltWith data we were able to identify 741 out of 778 package-repository matches. As with the Census data, the remaining 37 unmatched packages (less than 5%) are dropped from our analysis because they had been deleted from GitHub.

GHTorrent database. To obtain measures of the dispersion of the OSS value creation, we utilize the GHTorrent database, which contains the entire activity history on GitHub using GitHub's Representational State Transfer (REST) application programming interface (API). We leveraged its records of GitHub repositories, developer-level commits, and developer public profile information to estimate the contribution of each developer. We narrowed the sample for our developer contribution analysis in two steps. First, we winnowed the GitHub repositories and their commits from GHTorrent based on the repository URLs of our joint Census-BuiltWith sample.[10] Second, we focus on human contributions to OSS by removing approximately eight thousand (12%) GitHub contributors that we considered to be robots.[11] The final sample contains around sixty thousand developers and 2.3 million commits.

To prepare these three datasets for estimation, we first identified the number of lines of code and the programming languages used for each package using the OSS packages pygount (to count the number of lines of code) and linguist (to identify programming languages).[12] We categorize each distinct language into one of three different buckets moving from more likely human-written to more likely machine-written (see Table A1). Bucket 1 contains programming

[9] Since the package and package manager names are missing, a precise match using libraries.io was not feasible. Technology names are product names intended for customers and can be less technical and precise than the package names for internal development purposes. Thus, using technology names in a libraries.io search could cause significant ambiguity.

[10] The match rate is over 96%, with the unmatched repositories accounting for only 0.15% of our calculated OSS demand value discussed below.

[11] The filtering is based on the "fake user" classification by GHTorrent, as well as any usernames containing words "bot" or "robot", surrounded by special characters. This method is more conservative than other methods of bot detection like that used in the GitHub Innovation Graph (https://innovationgraph.github.com/), which rely on a monthly commit frequency threshold.

[12] https://pypi.org/project/pygount/ and https://github.com/github-linguist/linguist

and markup languages (which are most likely to be human-written), bucket 3 contains data (most likely to be machine-written), and bucket 2 contains anything in between, such as config files and batch processing (which are sometimes human-written and sometimes machine-written, but it is difficult to tell based purely on looking at the code).[13] For our primary estimation we only use bucket 1 while providing robustness checks for buckets 2 and 3 in the Appendix, and thus our results represent a lower bound. Finally, for some analyses we dig deeper and show the top 5 programming languages (as classified by GitHub, 2022 for the year 2020; the year our data is from). The top 5 programming languages contain C (including C# and C++), Java, JavaScript, Python, and Typescript. We also add Go to this list of top languages since it is an increasingly widely used language in OSS.

Similarly, to assess the economic significance of AI-related OSS, we categorize open source packages into a three-tiered hierarchy based on their functional proximity to AI production. Tier 1 represents core AI packages dedicated to model definition and training (OSS packages that are AI). Tier 2 encompasses the primitives that are required for AI development, such as linear algebra and tensor manipulation libraries (AI cannot exist without them, but they can exist without AI). Tier 3 includes packages that are needed to support the broader AI infrastructure, such as cloud and hardware integration tools. We utilized OpenAI's GPT 5.2 to classify these packages and applied the below described labor-market approach to estimate value, calculating the costs associated with the labor supply required to produce the software and the subsequent labor demand generated by its implementation.

3. Methodology

We first measure the value of OSS by considering the supply and the demand side of OSS using a labor market approach.[14] The thought experiment is that we live in a world where OSS does not exist and has to be recreated at each firm that uses a given piece of OSS. Using the labor market approach, we calculate the labor replacement cost of each OSS package. To estimate the value for each package, we use COCOMO II (Boehm, 1984; Boehm et al., 2009) at the individual package

[13] Note that our focus is the cost to recreate OSS code written by humans, not robots. However, directly removing all OSS contributions from robot accounts is infeasible here, because we cannot observe the exact lines of code written by the robot accounts on GitHub.

[14] In an alternative version, we use a mixture of a labor and goods market approach that is more closely aligned with Greenstein and Nagle (2014) and Murciano-Goroff, et al (2021). However, applying this method in our setting requires numerous additional assumptions due to data constraints. For simplicity we call it the goods market approach for which we provide the details and statistics, as well as the limitations in Appendix A.

level and then sum across all package values to obtain a supply-side labor market replacement value. Then, we scale the supply-side value by the number of times firms are using each package while removing multi-usage within each firm to obtain a demand-side value.

Second, we move beyond the aggregate and inspect inequality in the value creation process. In OSS, as with many creative endeavors, it is common that a small handful of individuals provide the bulk of the contributions, while many others make small contributions (sometimes referred to as the 80/20 rule, implying 80% of the work is done by 20% of the people). Research has shown these frequent contributors often attain positions of influence in OSS communities as a result of their efforts (Hanisch, et al, 2018). Therefore, to better understand the dispersion of value creation across developers, we first use the GHTorrent data and identify individual developer contributions in two ways: a) through their OSS value contributions directly and b) through the total number of repositories they contributed to. We then test how concentrated these two contribution measures are to better understand whether many or few developers contribute to the total value we measure. We explain the exact details for the labor market approach and the inequality of valuations below.

3.1 Labor market approach

For the labor market approach, we estimate the value of OSS by calculating the replacement value of a package. We ask how much it would cost to reproduce the package by hiring a programmer and paying them a competitive market wage. To estimate this supply-side value (V_S^{Labor}), we take the complete list of OSS packages discussed above, and then count the lines of code in each unique package.[15] For each unique package, *i*, we calculate the value and then sum over all these values to obtain the total value:

$$V_S^{Labor} = \sum_{i=1}^{N} V_{S_i} = \sum_{i=1}^{N} P_i * 1 \,. \qquad (1)$$

In this calculation, we implicitly do not incorporate any production externalities since we assume that there is no spillover knowledge from one package to the next that would lower the

[15] We use only the pure lines of code excluding documentation lines and empty lines. Hence, we underestimate the true value of recreating each package.

cost of programming.[16] This methodology is similar to that used by other papers that estimate the supply-side costs of OSS (Blind et al., 2021; Nagle, 2019b; Robbins et al., 2021).

We then calculate the demand-side value of OSS by incorporating the usage information (*Q)* for each package:

$$V_D^{Labor} = \sum_{i=1}^{N} V_{D_i} = \sum_{i=1}^{N} P_i * Q_i \,. \quad (2)$$

Here, we do not incorporate consumption externalities, i.e., we do not allow a benefit to arise for the general public when a package has been created and we further make sure that each firm is only replacing a package they use once, since a replaced package can be used within a firm as a club good (e.g., see Cornes and Sandler, 1996). The value V_S^{Labor} reflects the cost to rewrite all widely-used OSS once (e.g., the concept of OSS still exists, but all of these packages needed to be rewritten from scratch), while the value V_D^{Labor} reflects the cost for each firm that uses one of these OSS packages to pay a developer to rewrite those packages (e.g., OSS itself no longer exists).

For both the supply and demand models, we obtain the dollar-value for each package (P_i) via the Constructive Cost Model II, also abbreviated as COCOMO II (Boehm 1984, Boehm et al. 2009). The model has previously been used by the United States Department of Defense to estimate software project costs as well as in prior research estimating the value of OSS (Blind et al., 2021; Nagle, 2019b; Robbins et al., 2021). It is a highly flexible model that allows us to create non-linear transformations of lines of code to dollar-values. It uses the following modelling equation:

$$E_i = \alpha \eta L_i^{\beta} \quad (3)$$

where L represents the lines of codes in thousands and E the effort in person-month units for each project i. Consistent with Blind et al. (2021), we use the default parameter values for α, β and η. The parameters α, β are non-linear adjustment factors set to 2.94, and 1.0997 respectively. The parameter η is an effort-adjustment factor which can be modified to incorporate subjective assessments of product, hardware, personnel, and project attributes. Since we do not have a prior

[16] Similar to a representative agent model (e.g. see Williamson 2006), one may think of each package being reproduced by separate programmers who are identical replicas of each other (and therefore have the same skill level, but do not become more efficient via learning).

for each project, we set η to the default value of one. To obtain the price (P_i) of each OSS project, we then multiply the results of equation (3) by the weighted global wage that an average programmer would obtain on the open market. To calculate a global wage, we include the base monthly salaries of software developers from Salary Expert for the top 30 countries in terms of their GitHub developer counts in 2021 (Wachs et al., 2022).[17] The weight of each country is its share of active GitHub contributors over the total contributors in the top 30 countries. We further create bounds by using a low-wage (India) and high-wage (USA) labor market for the wage to better understand how the value would vary by the pool of programmers used to recreate all of OSS.[18]

3.2 Contribution Measurement

To better understand how value is created and whether it is created equally or unequally, we build up a graphical depiction in three steps. In the first step, we measure the value contributions by developers. In a second step, we obtain a measure for the number of repositories developers contribute to. Finally, we provide a graphical representation of both using the commonly known concept of the Lorenz curve to better understand the extent of inequality in contributions. We describe the details below.

Value Contribution. We calculated the supply and demand values of OSS that each developer contributed. At the repository level, we quantified each developer's proportional work contribution by calculating their share of commits to the total number of commits for a repository. This share was subsequently multiplied with the repository's demand and supply values separately to derive the value-added contribution of that individual contributor to the repository. Finally, we aggregate the value contributions across all repositories for each developer. The individual value contribution from a unique developer *Dev*, V_j^{Dev}, can be expressed as:

$$V_j^{Dev} = \sum_i^N \sigma_i^{Dev} * V_{ij} \quad (7)$$

[17] There are 179 counties in Wachs et al. (2022), but the top 30 countries consist of over 88% of the global active contributors with each of the rest having less than 0.6% share. The top 30 countries are listed in Table A2.

[18] We choose the high- and low-wage reference countries based on a combination of the number of active GitHub developers and the average annual software developer base wage.

where σ_i^{Dev} is the share of commits the focal developer made in repository *i*, and V_{ij} is the demand or supply value of the entire repository *i* specified in Equations (1) and (2), with $j \in \{D, S\}$, and N is the number of repositories in our main sample, i.e. Census and BuiltWith combined.

Repository Contribution. This is simply the number of repositories a given developer contributes to, and it is expressed as follows:

$$N^{Dev} = \sum_i^N \mathbb{1}\{\sigma_i^{Dev} > 0\} \quad (8)$$

where $\mathbb{1}$ is the indicator function equal to 1 when the developer has a non-zero number of commits to repository *i*. This measure entails the variety of OSS needs being addressed by individual developers. Jointly with the value contribution measure, they help us understand whether the value that is being generated overall is concentrated within a small number of developers. It may be generally more desirable for the whole OSS ecosystem and its diversity if individual developers participate in many repositories and not just a few.

Measuring the Dispersion of Contributions. To graphically examine the dispersion of developer contribution values, we utilized Lorenz curves (Lorenz, 1905), with respect to both demand and supply side values. Lorenz curves are a well-established way to represent inequality and, as such, they allow us to better understand how dispersed developer contributions to OSS within the private economy are. Developers are systematically arranged in ascending order based on their contributions to OSS demand and supply, as delineated in Equation (7). Subsequently, these ranks were normalized to a scale ranging from 0 to 100 percentiles, serving as the x-axis values for the Lorenz Curves. The y-axis, on the other hand, presents the corresponding value contributions V_j^{Dev}. The graphical representation in the results section will elucidate the degree of inequality pertaining to the value contributions among developers. To supplement the analysis, we further investigated how dispersed the repository contribution, N^{Dev}, is by plotting Equation (8). This enables us to ascertain whether any substantial value inequality stems from top contributors predominantly focusing on a narrow subset of exceptionally popular repositories, or alternatively, from their engagement with a broader spectrum of successful repositories.

4. Results

After applying the labor market approach using COCOMO II, we obtain global estimates for the value of OSS. To calculate the overall value, we first need the underlying number of lines of code (to calculate the supply-side value) and then the usage statistics (for the demand-side value). Since there may be substantial heterogeneity in value by programming languages, we also show the top programming languages during our investigation period in the year 2020, as discussed above.

--- Table 1 about here ---

Table 1 shows descriptive statistics from both data sets separately. The inward-facing Census (Panel A) contains just over 261.7 million lines of code with 72% of the lines being attributable to the top programming languages. The average package includes 142 thousand lines of code with a slightly lower average of 113 thousand lines of code for the subset of top languages. When considering the demand (usage) side, we observe that the Census packages were used over 2.7 million times while 92% of this usage is attributable to the top languages. The average package was used 1,472.4 times with a higher usage of around 1,497.5 for the top languages. For the outward-facing BuiltWith data we find similar patterns, but at different levels. The packages included from BuiltWith include over 82 million lines of code, 71% of which are attributable to the top programming languages. The average package within the BuiltWith sample has 111 thousand lines of code with a lower average of around 80 thousand lines of code for the top languages. This is because our BuiltWith data primarily consists of JavaScript-based packages, which are often smaller than packages written in other languages. BuiltWith packages were used over 142 thousand times where 99.97% were attributable to the top languages. Next, we used these raw observations to calculate the value of all OSS through the labor market approach and estimated the value created from the supply and demand side.

--- Table 2 about here ---

Table 2 shows the estimates for the value of OSS based on the firm-relevant joint Census and BuiltWith sample. All estimates in Table 2 are based only on software languages classified in bucket 1 from Table A1, which are the most likely to be written by a human rather than a

machine.[19] The first column contains estimates with wages from a low income country (India), the global average wage, and a high income country (United States of America), respectively (as described above). To reproduce all widely-used OSS once (e.g., the idea of OSS still exists, but all current OSS is deleted and needs to be coded from scratch), using programmers at the average developer wage from India, it would require an investment of $1.22 billion. In contrast, if we use the average developer wage from the United States, then reproducing all widely-used OSS would require an investment of $6.22 billion. Using a pool of programmers from across the world, weighed based on the existing geographic contributions to OSS as discussed above, would lead to an investment somewhere in between the low and high-income country, $4.15 billion. It is useful to compare these numbers to those from similar studies to understand differences in valuing all OSS (prior studies) versus that which is widely used (our study). Robbins et al. (2021) and Blind et al. (2021) use a method similar to ours and estimate that the value of OSS created in the US is $38 billion in 2019 and that created in the EU is €1 billion in 2018. Wachs et al. (2022) show that roughly 50% of OSS contributions come from the US and EU combined. In aggregate, that would lead these studies to give a global value of OSS of $78 billion. Thus, our middle estimate of a supply-side value of $4.15 billion for only firm-oriented and widely used OSS is a credible lower-bound and highlights the higher estimations of the total of supply-side value of OSS when not considering whether or not a given OSS package is widely used. This further indicates that the supply-side value of the most widely used OSS is roughly 5.5% of the supply-side value of all OSS.

The second column in Table 2 contains the demand side estimates based on the labor market approach. We find that if firms had to recreate all OSS packages they used (e.g., OSS itself no longer existed and every firm that used an OSS package had to recreate it), then the entire cost would amount to between $2.59 trillion to $13.18 trillion using labor from the low wage or high wage country only, respectively. A pool of programmers across the globe could recreate all of OSS that is being widely used for a cost of approximately $8.80 trillion. Interpreting this number is slightly more complicated but still feasible. According to a Statista (2023) report, global software revenue in 2020 (the same year as our data) was $531.7 billion. However, this represents a flow, not a stock, of software. Relying on government estimates that software fully depreciates over

[19] Tables in the appendix show the equivalent values from Table 2 when including software language buckets 1 and 2 (Table A3) and all three software language buckets (Table A4).

three years, we can do a back of the envelope calculation and consider the purchase value of the full stock of prepackaged software used in 2020 as the aggregate of that sold from 2018 to 2020, which is $1.54 trillion. Further, this represents only the expenditure on prepackaged software and does not include custom purchased or in-house developed software. Here, the best obtainable estimates of private-sector investment in software overall come from the United States National Income and Product Accounts data (NIPA 2023). In 2020, the national account data shows that in the US, private firms spent $479.2 billion on software, of which 45% ($215.5 billion) was prepackaged. If we assume this is a consistent ratio for the rest of the world, then the total amount firms spent on software being used in 2020 was $3.4 trillion (= $1.54 trillion/0.45). Combining this rough estimate with the demand side estimate of the value of OSS based on an average global wage ($8.8 trillion), this indicates firms would spend $12.2 trillion (=$3.4 trillion + $8.8 trillion), or **three and a half times** what they currently spend if they needed to pay in-house developers to write the OSS that they currently use for free.

--- Figure 1 about here ---

Figure 1 shows the heterogeneity of the value of OSS across the top programming languages. Panel A shows the supply side value with the labor value being displayed on the vertical axis. We find that OSS packages created in Go have the highest value with $803 million in value that would have to be created from scratch if the OSS packages did not exist. Go is closely followed by JavaScript and Java with $758 million and $658 million, respectively. The value of C and Typescript is $406 million and $317 million, respectively, while Python has the lowest value of the top languages with around $55 million. JavaScript is not only the top language on GitHub since at least 2014 (GitHub, 2022) it is also the language with one of the highest values in our data. In contrast, Python became more popular over time moving up from the number four spot to the number two language being used in 2020 across all OSS packages on GitHub, while it is in the last spot of our top languages.

Panel B shows the demand side value across the top programming languages. Based on usage generated value, Go is more than four times the value of the next language, JavaScript. Typescript (a language that extends JavaScript) has seen immense growth rising from the tenth spot of the top 10 languages in 2017 to the fourth spot in 2020 which is also reflected in our data

with Typescript being the third most important language on the demand side. The two web languages are followed by C and far behind are Java, and Python.

--- Figure 2 about here ---

Figure 2 splits the OSS value estimates for each language by our inward facing (Census) and outward facing (BuiltWith) data sources. Panel A and Panel B focus on the supply and demand side estimation for the Census. We obtain a similar pattern that has been established already in the aggregate when pooling both data sources (Figure 1) albeit the impact of JavaScript is substantially lower. On the supply side of the Census in Panel A, Java has the second-highest value while JavaScript code from the Census contributes to a substantially lower value to the aggregate. Similarly, the supply side values of C, Python, and Typescript are mainly driven by the Census. From the demand side Panel B we find that Go is the most popular language for inward facing code while all other languages appear to be negligible in relative terms.

Figure 2, Panel C and Panel D show the supply and demand side values for the BuiltWith dataset. The supply side value in Panel C clearly indicates that the value from the BuiltWith sample is driven by JavaScript code, which is a good sanity check since we focused on JavaScript packages to proxy for OSS in the BuiltWith sample, as discussed above. The second highest value is created by TypeScript which is reassuring since it is a superset of JavaScript. Panel D shows a similar pattern on the usage side where most of the value arises from JavaScript while TypeScript is trailing at the second spot as well. The other languages contribute only marginal amounts to the supply and demand side values. Overall, these findings are broadly consistent with the main use cases of the various languages (web programming vs. application programming) and an idea that languages through which value are generated are not necessarily identical to languages that are used by the general public.

--- Figure 3 about here ---

Figure 3 shows the demand-side value of OSS across industries by NAICS 2-digit codes using the BuiltWith online data.[20] This can be interpreted as the value each of these industries

[20] Due to the Census containing proprietary customer information, it did not reveal industries across the whole dataset and as such we can only show the value across industries using the outward-facing data from BuiltWith. We match BuiltWith websites to industry information from the Orbis and Compustat datasets. 94.6% of BuiltWith websites are

receive because OSS exists. The industry with the highest usage value of around $43 billion is "Professional, Scientific, and Technical Services." "Retail Trade" as well as "Administrative and Support and Waste Management and Remediation Services" make up another large part of the demand-side externally facing value of OSS with $36 billion and $35 billion, respectively. In contrast, industries that constitute just a small portion of the value are "Mining, Quarrying, and Oil and Gas Extraction", "Utilities", "Agriculture, Forestry, Fishing, and Hunting." The latter industries are classical non-service sector industries and as such software is expected to play less of a role there.

-- Figure 4 about here ---

Figure 4 shows the Lorenz curves and the number of repositories that a fraction of programmers has contributed to for the supply side (Panel A) and the demand side (Panel B). A Lorenz curve that lay directly on the 45 degree line would imply a very even distribution of values across programmers. Instead, Panel A shows a Lorenz curve as a nearly flat line with a drastic increase for the final share of programmers. This implies that the distribution of the supply value is highly uneven and considerably more concentrated than the 80/20 standard. Indeed, the last five percent of programmers, or 3,000 programmers, generate over 93% of the supply side value. Similarly, Panel B shows – when accounting for usage – that those last five percent generate over 96% of the demand side value. In aggregate, this indicates that a very small number of programmers are creating the bulk of OSS code that is heavily relied upon by firms to create their own code. In both Panel A and Panel B we can also see a rise in the number of repositories for the last 10-15% of programmers that contribute to the highest value, which implies that the uneven value that is generated by few programmers is not just due to a few highly valuable repositories but by the contributions of this handful of contributors to a substantial number of repositories.

--- Table 3 about here ---

Table 3 shows the estimates for the value of open source software packages that have the potential to contribute to the production of products and services that use artificial intelligence

matched with an industry through this process. For firms (domains) that are associated with multiple industries we took the average value and distributed it across industries.

based on the global wage scenario from Table 2. We find a staggering disparity between the supply-side costs and the demand-side value, particularly within Tier 2 (AI primitive OSS packages). While the labor required to create these packages represents a significant investment, the labor demand they support in the broader economy reaches into trillions of dollars. The value estimates shown under Tiers 1 and 2 indicate a lower bound of the value of artificial intelligence within the 2020 data, which predates the explosion of AI seen with the release of ChatGPT in late 2022. Although Tier 3 can have much broader scope of usage context and therefore exhibits the greatest demand and supply values, the findings together indicate how prepared firms were to benefit from the AI revolution by leveraging existing infrastructure and mathematical foundations. These figures suggest that even before generative AI became a household term, the open source ecosystem had already established a trillion-dollar foundation for a future industrial transformation.

5. Conclusion

In this study we estimate the value of widely-used open source software globally with two unique datasets: the Census of OSS and the BuiltWith data. We are able to estimate not only the supply side value of existing code (e.g., the cost it would take to rewrite each piece of widely-used OSS once) but also the demand side value for the private economy (e.g., the cost it would take for each company that uses a piece of OSS to rewrite it). While we do not focus on the long tail of OSS, we consider this an additional contribution of our study as focusing on OSS that is widely used allows us to more precisely understand the value created by OSS, rather than only measuring the replacement cost for all OSS (which would overestimate the true value since many OSS projects are not used in production code). However, although we highlight the substantial value that OSS has in our society based on a wide swath of usage data, it is not feasible to identify 100% of the OSS used across the world and, as such, our demand-side estimates are likely an underestimate of the true value.

Adjusted for usage, we find a large demand-side value of OSS of $8.8 trillion when using programmers from across the world, with some variance, depending on whether we would hire programmers from a low- or high-income country only. There is substantial heterogeneity in the value across programming languages and whether the code is inward–facing – i.e., for creating products that are being sold – or outward-facing – i.e., used on the company's website. The top 6 programming languages create 84% of the demand-side value. We also show substantial

heterogeneity by industries and, finally, heterogeneity in the value contributions by programmers themselves. Over 95% percent of the demand-side value is generated by only five percent of programmers, and those programmers contribute not only to a few widely used projects but to substantially more projects than the programmers that are engaged at the lower end of the value distribution. Further, the open source software ecosystem already supported a staggering value of $1.46 trillion for the development of AI models prior to the release of Chat-GPT.

In aggregate, our results show the substantial value that OSS contributes to the economy despite this value generally showing up as zero via direct measurement since prices equal zero and quantity is difficult to measure using public data alone. It also shows a unique way of measuring the value of artificial intelligence through the lens of open source software. As such, our research lays the groundwork for future studies of not only OSS, but of all IT, including artificial intelligence, and its growing impact on the global economy.

References

Almeida, D. A., Murphy, G. C., Wilson, G., & Hoye, M. (2017, May). Do software developers understand open source licenses? In 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC) (pp. 1-11). IEEE.

Andreessen, M. (2011). Why Software Is Eating The World. Accessed May 1, 2023. Source: https://www.wsj.com/articles/SB10001424053111903480904576512250915629460

Aristotle. (1981). Politics. Book 2, Chapter 3. T.A. Sinclair translation. Penguin Books, London.

Blind, K., Böhm, M., Grzegorzewska, P., Katz, A., Muto, S., Pätsch, S., & Schubert, T. (2021). The impact of Open Source Software and Hardware on technological independence, competitiveness and innovation in the EU economy. European Commission, Ed.

Blind, K., & Schubert, T. (2023). Estimating the GDP effect of Open Source Software and its complementarities with R&D and patents: evidence and policy implications. The Journal of Technology Transfer, 1-26.

Boehm, B. W. (1984). Software engineering economics. IEEE transactions on Software Engineering, (1), 4-21.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2009). Software cost estimation with COCOMO II. Prentice Hall Press.

Branstetter, Lee G., Matej Drev, and Namho Kwon. (2019). "Get with the program: Software-driven innovation in traditional manufacturing." Management Science 65, no. 2: 541-558.

Brynjolfsson, E. (1993). The productivity paradox of information technology. Communications of the ACM, 36(12), 66-77.

Brynjolfsson, E., & Hitt, L. (1996). Paradox lost? Firm-level evidence on the returns to information systems spending. Management Science, 42(4), 541-558.

Brynjolfsson, E., Rock, D., & Syverson, C. (2018). Artificial intelligence and the modern productivity paradox: A clash of expectations and statistics. In The economics of artificial intelligence: An agenda (pp. 23-57). University of Chicago Press.

Burton, R. M., Håkonsson, D. D., Nickerson, J., Puranam, P., Workiewicz, M., & Zenger, T. (2017). GitHub: exploring the space between boss-less and hierarchical forms of organizing. *Journal of Organization Design*, *6*, 1-19.

Conti, A., Peukert, C., & Roche, M. |(2023). "Beefing IT up for your Investor? Open Sourcing and Startup Funding: Evidence from GitHub." *Harvard Business School Working paper No. 22-001*.

Cornes, R., & Sandler, T. (1996). The theory of externalities, public goods, and club goods. Cambridge University Press.

DeStefano, T., and J. Timmis (2023). Demand Shocks and Data Analytics Diffusion, working paper.

Dushnitsky, G., & Stroube, B. K. (2021). Low-code entrepreneurship: Shopify and the alternative path to growth. Journal of Business Venturing Insights, 16, e00251.

Eisfeldt, A. L., & Papanikolaou, D. (2014). The value and ownership of intangible capital. American Economic Review, 104(5), 189-194.

European Commission. (2020). Open Source Software Strategy 2020-2023. Luxembourg: Office for Official Publications of the European Communities.

Executive Order No. 14028. (2021). Executive Order on Improving the Nation's Cybersecurity. May 2021.

Fackler, T., Hofmann, M., & Laurentsyeva, N. (2023). *Defying Gravity: What Drives Productivity in Remote Teams?* (No. 427). CRC TRR 190 Rationality and Competition.

Greenstein, S., & Nagle, F. (2014). Digital dark matter and the economic contribution of Apache. *Research Policy*, *43*(4), 623-631.

GitHub (2022). "Octoverse: The state of open source software." Accessed November 3, 2023. https://octoverse.github.com/2022/top-programming-languages.

Hanisch, M., Haeussler, C., Berreiter, S., & Apel, S. (2018, July). Developers' progression from periphery to core in the Linux kernel development project. In *Academy of Management Proceedings* (Vol. 2018, No. 1, p. 14263). Briarcliff Manor, NY 10510: Academy of Management.

Hardin, G. (1968). "The Tragedy of the Commons". Science. 162 (3859): 1243–1248.

Henkel, J. (2009). Champions of revealing—the role of open source developers in commercial firms. Industrial and Corporate Change, 18(3), 435-471.

Jacks, J. (2022). Open Source Is Eating Software FASTER than Software Is Eating The World. Accessed May 1, 2023. Source: https://www.coss.community/cossc/open-source-is-eating-software-faster-than-software-is-eating-the-world-3b01

Kim, D. Y. (2020). Product Market Performance and Openness: The Moderating Role of Customer Heterogeneity. In *Academy of Management Proceedings* (Vol. 2020, No. 1, p. 21309). Briarcliff Manor, NY 10510: Academy of Management.

Koning, R., Hasan, S., & Chatterji, A. (2022). Experimentation and start-up performance: Evidence from A/B testing. Management Science, 68(9), 6434-6453.

Krishnan, M. S., Kriebel, C. H., Kekre, S., & Mukhopadhyay, T. (2000). An empirical analysis of productivity and quality in software products. Management science, 46(6), 745-759.

Lerner, J., & Tirole, J. (2005). The scope of open source licensing. Journal of Law, Economics, and Organization, 21(1), 20-56.

Lorenz, M. O. (1905). "Methods of measuring the concentration of wealth". Publications of the American Statistical Association. Publications of the *American Statistical Association*, Vol. 9, No. 70. 9 (70): 209–219. Bibcode:1905PAmSA...9..209L. doi:10.2307/2276207. JSTOR 2276207.

Lifshitz-Assaf, H., & Nagle, F. (2021). The digital economy runs on open source. Here's how to protect it. Harvard Business Review Digital Articles. https://hbr.org/2021/09/the-digital-economy-runs-on-open-source-heres-how-to-protect-it.

Lloyd, W. F. (1833). Two lectures on the checks to population: Delivered before the University of Oxford, in Michaelmas Term 1832. JH Parker.

Maracke, C. (2019). Free and Open Source Software and FRAND-based patent licenses: How to mediate between Standard Essential Patent and Free and Open Source Software. The Journal of World Intellectual Property, 22(3-4), 78-102.

Murciano-Goroff, R., Zhuo, R., & Greenstein, S. (2021). Hidden software and veiled value creation: Illustrations from server software usage. *Research Policy*, *50*(9), 104333.

Musseau, J., Meyers, J. S., Sieniawski, G. P., Thompson, C. A., & German, D. (2022, May). Is open source eating the world's software? Measuring the proportion of open source in proprietary software using Java binaries. In *Proceedings of the 19th International Conference on Mining Software Repositories* (pp. 561-565).

Nagle, F. (2018). Learning by contributing: Gaining competitive advantage through contribution to crowdsourced public goods. Organization Science, 29(4), 569-587.

Nagle, F. (2019a). Open source software and firm productivity. Management Science, 65(3), 1191-1215.

Nagle, Frank (2019b). "Government Technology Policy, Social Value, and National Competitiveness." Harvard Business School Working Paper, No. 19-103, March 2019.

Nagle, F., Dana, J., Hoffman, J., Randazzo, S., & Zhou, Y. (2022). Census II of Free and Open Source Software—Application Libraries. *Linux Foundation, Harvard Laboratory for Innovation Science (LISH) and Open Source Security Foundation (OpenSSF). https://www.linuxfoundation.org/research/census-ii-of-free-and-open-source-software-application-libraries.*

NIPA (2023). Bureau of Eonomic Analysis, NIPA Table 5.6.5. accessed: 2023-11-14, source: https://apps.bea.gov/iTable/?reqid=19&step=3&isuri=1&select_all_years=0&nipa_table_list=331&series=q&first_year=2013&last_year=2023&scale=-9.

Nordhaus, William D., 2006, "Principles of National Accounting for Nonmarket Accounts," in *A New Architecture for the US National Accounts*, editors, Dale W. Jorgenson, J. Steven Landefeld, and William D. Nordhaus, University of Chicago Press.

Ostrom, Elinor (1990). Governing the commons: The evolution of institutions for collective action. Cambridge: Cambridge University Press.

Peters, R. H., & Taylor, L. A. (2017). Intangible capital and the investment-q relation. Journal of Financial Economics, 123(2), 251-272.

Robbins, C., Korkmaz, G., Guci, L., Calderón, J. B. S., & Kramer, B. (2021). A First Look at Open-Source Software Investment in the United States and in Other Countries, 2009-2019.

Singh, Shivendu Pratap (2020) *Products, Platforms, and Open Innovation: Three Essays on Technology Innovation.* Doctoral Dissertation, University of Pittsburgh. (Unpublished)

Solow, R. (1987). "We Better Watch Out." *New York Times Book Review*, July 1987, p. 36.

Statista (2023). Statista Software Worldwide, accessed 2023-11-14, source: - https://www.statista.com/outlook/tmo/software/worldwide#revenue, accessed November 2023.

Synopsys (2023). 2023 OSSRA: A deep dive into open source trends. Accessed May 1, 2023. Source : https://www.synopsys.com/blogs/software-security/open-source-trends-ossra-report/

Tang, S., Wang, Z., & Tong, T. (2023). Knowledge Governance in Open Source Contributions: The Role of Gatekeepers. In *Academy of Management Proceedings* (Vol. 2023, No. 1, p. 17622). Briarcliff Manor, NY 10510: Academy of Management.

Tozzi, C. (2016). "Open Source History: Why Did Linux Succeed?" *Channel Futures*, August, 2016. Accessed November 3, 2023. https://www.channelfutures.com/open-source/open-source-history-why-did-linux-succeed

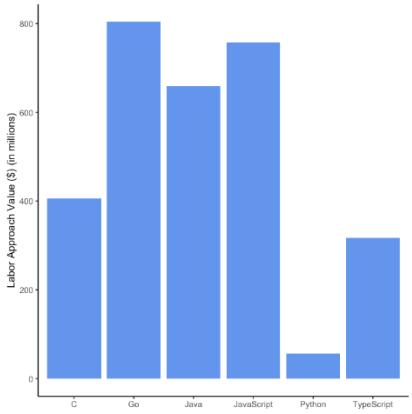
Wachs, J., Nitecki, M., Schueller, W., & Polleres, A. (2022). The geography of open source software: Evidence from github. *Technological Forecasting and Social Change*, *176*, 121478.

Williamson, S. (2006). Notes on macroeconomic theory. University in St. Louis. Department of Economics.

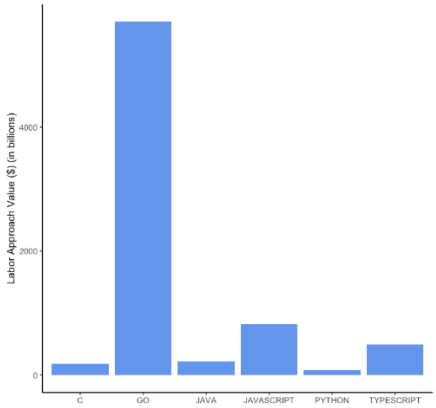
Zhang, Y., Zhou, M., Mockus, A., & Jin, Z. (2019). Companies' participation in oss development–an empirical study of openstack. IEEE Transactions on Software Engineering, 47(10), 2242-2259.

Figure 1 The value of open source software: top languages

Panel A. Supply Side



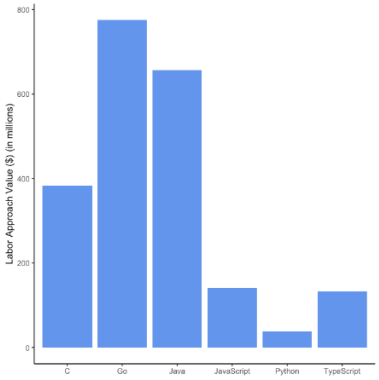
Panel B. Demand Side



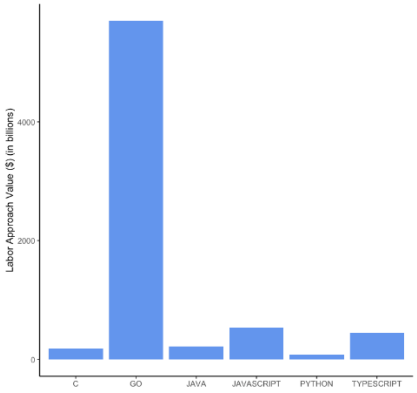
Note. The figures show the labor market value for the top-5 languages according to GitHub plus Go. Panel A displays the supply side while Panel B incorporates usage. On the labor side we use our estimated average global wage for programmers as explained in the methodology section.

Figure 2 The value of open source across top-5 languages + Go and across data sources

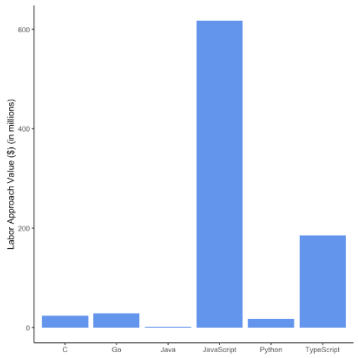
Panel A. Census. Supply Side



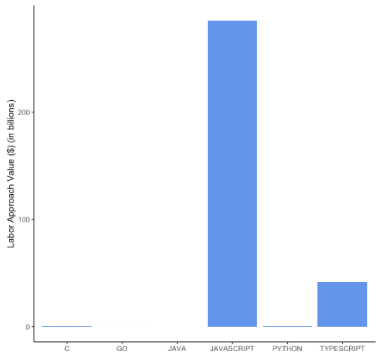
Panel B. Census. Demand Side



Panel C. BuiltWith. Supply Side

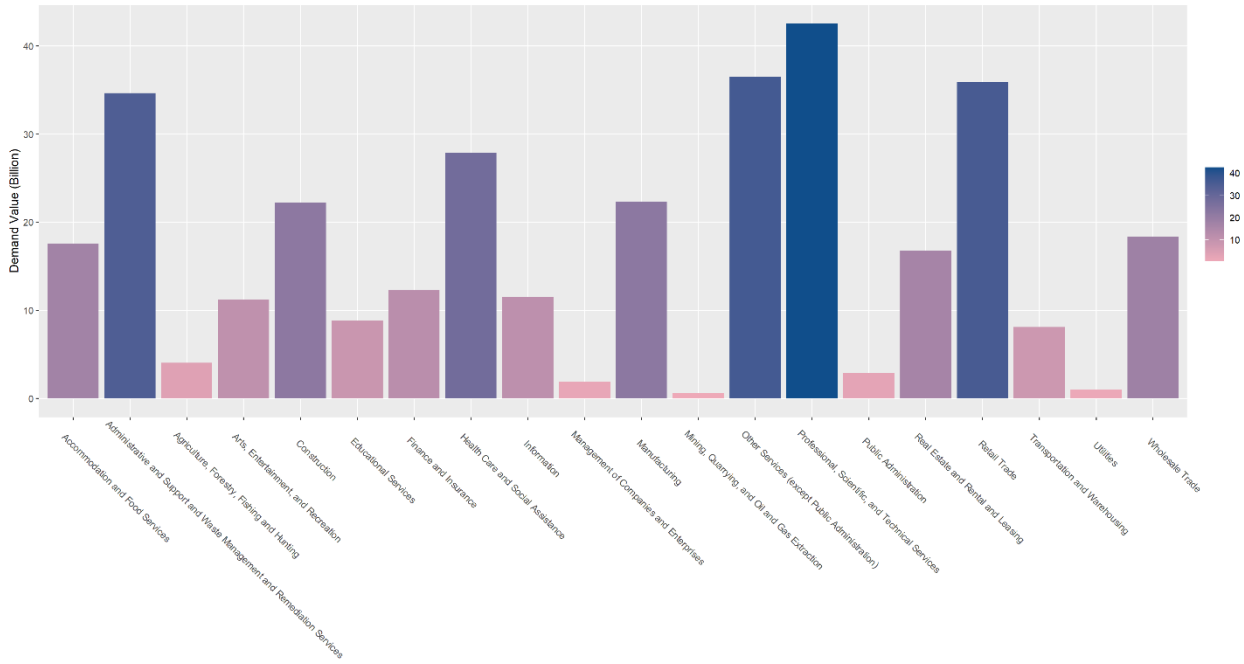


Panel D. BuiltWith. Demand Side



Note. The figures show the labor and goods market value for the top-5 languages (according to GitHub) + Go split by inward facing (Census) and outward facing (BuiltWith) data source. Panel A and Panel B show the supply side and demand side values for the Census and Panels C and D the supply and demand side values for BuiltWith.

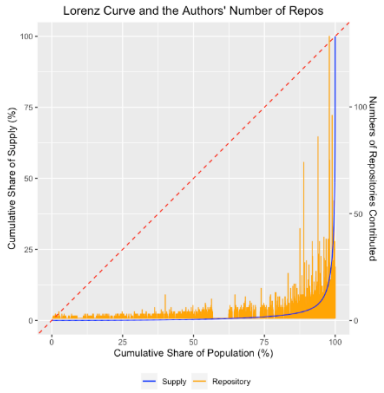
Figure 3 The value of open source across industries from websites



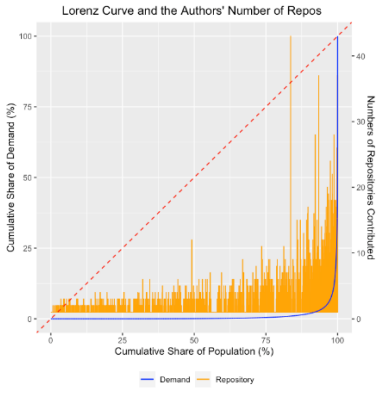
Note. The figure shows the demand side labor value across NAICS 2-digit code industries using the Built With data. For firms (domains) that are associated with multiple industries we took the average value and distributed it across industries.

Figure 4 The dispersion of the value creation of open source

Panel A. Supply Side



Panel B. Demand Side



Note. The figures show the Lorenz curve of the labor market value contribution per developer (in blue) as well as the number of repositories that a fraction of programmers contributed to (in yellow). Panel A displays the supply side while Panel B incorporates usage.

Table 1 Descriptive statistics on lines of code and usage

	Sum	Mean	SD	Obs
Panel A: Census				
Lines of Code – All packages	261,653,728	142,203.1	887,937.2	1,840
Lines of Code – Top 5 Languages and Go	189,673,184	113,712.9	702,832.1	1,668
Usage – All packages	2,709,155	1472.4	2,167.9	1,840
Usage – Top 5 Languages and Go	2,497,785	1497.5	2,228.8	1,668
Panel B: BuiltWith				
Lines of Code – All packages	82,504,613	111,342.3	613,488.1	741
Lines of Code – Top 5 Languages and Go	58,664,935	79,925.0	354,415.0	734
Usage – All packages	142,794.4	192.7	733.4	741
Usage – Top 5 Languages and Go	142,751.2	194.5	736.6	734

Note. The statistics are based on the lines of codes of different repositories. Panel A (B) portrays the aggregate sum, mean, standard deviation and number of observations for the Census (BuiltWith) data across lines of code and usage using all packages from bucket 1 (see Table A1).

Table 2 The labor-market value of open source software

	Labor Supply	Labor Demand
Wage: Low	$1.22 Billion	$2.59 Trillion
Wage: Global	$4.15 Billion	$8.80 Trillion
Wage: High	$6.22 Billion	$13.18 Trillion

Note. The high wage scenario is based on the US average wage and the low wage scenario is the Indian average wage for programmers in 2020. The global wage is an average wage from the countries in Table A4, weighted according to their contributions to OSS. These estimates include only languages from software classified in bucket 1 (see Table A1).

Table 3 The labor-market value of open source software packages with AI potential

	Labor Supply	Labor Demand
Panel A. All		
Tier 1	$0.45 Million	$3.37 Million
Tier 2	$60.52 Million	$1.46 Trillion
Tier 3	$1.20 Billion	$5,89 Trillion

Note. Tier 1 contains core artificial intelligence packages for model development, i.e. software whose primary function is the definition, training, or inference of probabilistic models. Tier 2 contains libraries that provide the mathematical primitives (linear algebra, tensor manipulation, statistical analysis) required for AI but are not AI themselves. Tier 3 contains artificial intelligence infrastructure packages, i.e. cloud and hardware. All packages have been classified using OpenAI's GPT 5.2 and the estimates have been calculated using the global wage scenario.

Online Appendix

Table A1 Languages within each bucket

Type	Language
Panel A: Bucket 1 - Languages	
Mark-Up Language	BIBTEX
Mark-Up Language	COLDFUSION HTML
Mark-Up Language	DOCBOOK XML
Mark-Up Language	HAML
Mark-Up Language	HTML
Mark-Up Language	HXML
Mark-Up Language	JAVAEE XML
Mark-Up Language	MARKDOWN
Mark-Up Language	MASON
Mark-Up Language	MXML
Mark-Up Language	RELAX-NG COMPACT
Mark-Up Language	RHTML
Mark-Up Language	TEX
Mark-Up Language	XML
Mark-Up Language	XQUERY
Mark-Up Language	YAML
Programming Language	ABNF
Programming Language	ACTIONSCRIPT
Programming Language	ADA
Programming Language	APPLESCRIPT
Programming Language	ARDUINO
Programming Language	ASPECTJ
Programming Language	ASPX-CS
Programming Language	ASPX-VB
Programming Language	AWK
Programming Language	C
Programming Language	C#
Programming Language	CHARMCI
Programming Language	CLOJURE
Programming Language	COFFEESCRIPT
Programming Language	COMMON LISP
Programming Language	CSS
Programming Language	CUDA
Programming Language	CYTHON
Programming Language	D
Programming Language	DART
Programming Language	DELPHI
Programming Language	EASYTRIEVE
Programming Language	EC
Programming Language	ELIXIR
Programming Language	ELM
Programming Language	EMACSLISP
Programming Language	ERLANG
Programming Language	F#
Programming Language	FISH
Programming Language	FORTH
Programming Language	FORTRAN
Programming Language	FORTRANFIXED

Programming Language	GAP
Programming Language	GHERKIN
Programming Language	GLSL
Programming Language	GO
Programming Language	GRAPHVIZ
Programming Language	GROOVY
Programming Language	HASKELL
Programming Language	HAXE
Programming Language	IDL
Programming Language	JAVA
Programming Language	JAVA SERVER PAGE
Programming Language	JAVASCRIPT
Programming Language	KOTLIN
Programming Language	LESSCSS
Programming Language	LIQUID
Programming Language	LIVESCRIPT
Programming Language	LLVM
Programming Language	LOGOS
Programming Language	LUA
Programming Language	MATHEMATICA
Programming Language	MINISCRIPT
Programming Language	MODULA-2
Programming Language	NASM
Programming Language	NIX
Programming Language	OBJECTIVE-C
Programming Language	OBJECTIVE-J
Programming Language	OCAML
Programming Language	OPENEDGE ABL
Programming Language	PAWN
Programming Language	PERL
Programming Language	PHP
Programming Language	PL/PGSQL
Programming Language	POSTSCRIPT
Programming Language	POVRAY
Programming Language	PROLOG
Programming Language	PROPERTIES
Programming Language	PUPPET
Programming Language	PYTHON
Programming Language	REASONML
Programming Language	REBOL
Programming Language	REDCODE
Programming Language	REXX
Programming Language	RUBY
Programming Language	RUST
Programming Language	S
Programming Language	SASS
Programming Language	SCALA
Programming Language	SCILAB
Programming Language	SCSS
Programming Language	SLIM
Programming Language	SMALLTALK
Programming Language	SOLIDITY
Programming Language	STANDARD ML
Programming Language	SWIFT
Programming Language	SWIG
Programming Language	TADS 3

Programming Language	TCL
Programming Language	THRIFT
Programming Language	TRANSACT-SQL
Programming Language	TREETOP
Programming Language	TYPESCRIPT
Programming Language	VB.NET
Programming Language	VBSCRIPT
Programming Language	VCL
Programming Language	VIML
Programming Language	WEB IDL

Panel B: Bucket 2 – Auxiliary Languages

Assembler/Compiler/Interpreter/Macro Processors	GAS
Assembler/Compiler/Interpreter/Macro Processors	M4
Assembler/Compiler/Interpreter/Macro Processors	RAGEL IN RUBY HOST
Configuration	CMAKE
Configuration	INI
Configuration	MAKEFILE
Configuration	NGINX CONFIGURATION FILE
Configuration	NSIS
Configuration	SQUIDCONF
Configuration	TERRAFORM
Configuration	TOML
Formatting	GROFF
IDE	NETBEANS PROJECT
Template Engine	CHEETAH
Template Engine	GENSHI
Template Engine	PUG
Template Engine	SMARTY
Template Engine	VELOCITY
Terminal/Batch	ANT
Terminal/Batch	APACHECONF
Terminal/Batch	BASH
Terminal/Batch	BATCHFILE
Terminal/Batch	MAVEN
Terminal/Batch	POWERSHELL
Terminal/Batch	RPMSPEC
Terminal/Batch	SINGULARITY
Terminal/Batch	TCSH
Translational	GETTEXT CATALOG

Panel C: Bucket 3 - Data

Data	BNF
Data	DIFF
Data	DTD
Data	E-MAIL
Data	JSON
Data	PROTOCOL BUFFER
Data	RESTRUCTUREDTEXT
Data	TEXT ONLY
Data	XSLT

Table A2 Top 30 countries included for the global wage

Country
United States
China
Germany
India
United Kingdom
Brazil
Russia
France
Canada
Japan
South Korea
Netherlands
Spain
Poland
Australia
Sweden
Ukraine
Italy
Switzerland
Indonesia
Taiwan
Colombia
Argentina
Mexico
Norway
Belgium
Denmark
Finland
Vietnam
Austria

Note. The top 30 countries are sorted in ascending order by GitHub user shares, and they include 88% of GitHub activity from 2020.

Table A3 The labor-market value of open source using languages in buckets 1 and 2

	Labor Supply	Labor Demand
Wage: Low	$1.23 Billion	$2.60 Trillion
Wage: Global	$4.18 Billion	$8.84 Trillion
Wage: High	$6.26 Billion	$13.24 Trillion

Note. The high wage scenario is based on the US average wage and the low wage scenario is the Indian average wage for programmers in 2020. The global wage is an average wage from the countries in Table A4. The estimates include only languages from buckets 1 and 2 (see Table A1).

Table A4 The labor-market value of open source using languages in buckets 1, 2, and 3

	Labor Supply	Labor Demand
Wage: Low	$1.88 Billion	$3.52 Trillion
Wage: Global	$6.41 Billion	$11.96 Trillion
Wage: High	$9.59 Billion	$17.91 Trillion

Note. The high wage scenario is based on the US average wage and the low wage scenario is the Indian average wage for programmers in 2020 The global wage is an average wage from the countries in Table A4. The estimates include only languages from buckets 1, 2, and 3 (see Table A1).

Table A5 Goods Basket – Equivalent Open Source and Proprietary Software

Open Source Software	Proprietary Software
Apache Http Server	Windows Server 2008
Audacity	Adobe Audition
Blender	Autodesk Maya
Elasticsearch	Amazon Kendra
FileZilla	SmartFTP
FreeCAD	AutoCAD
GIMP	Adobe Photoshop
GNU Octave	MATLAB
GnuCash	QuickBooks
KeePass	1Password
LibreOffice	Microsoft Office Suite
MariaDB server	Microsoft SQL Server
Metabase	Tableau
MySQL	Oracle MySQL
OpenVPN	ExpressVPN
PSPP	SPSS
Redis	Redis Enterprise
TensorFlow	TensorFlow Enterprise
VirtualBox	VMware Workstation
VLC Media Player	CyberLink PowerDVD

Table A6 The goods-market value of open source using languages in buckets 1, 2, and 3

	Goods-Demand Bucket 1	Goods-Demand Bucket 1-2	Goods Demand, Bucket 1-3
Wage: Low	$177 Million	$179 Million	$242 Million
Wage: Global	$177 Million	$179 Million	$242 Million
Wage: High	$177 Million	$179 Million	$242 Million

Note. The high wage scenario is based on the US average wage and the low wage scenario is the Indian average wage for programmers in 2020 The global wage is an average wage from the countries in Table A4. The estimates include only languages from buckets 1, 2, and 3 (see Table A1).

Appendix A) Goods-Market Valuation Approach

As an alternate estimation method, instead of using the labor replacement cost, we use a goods replacement value approach. We identify several OSS packages that have similar closed-source, pecuniary alternatives and consider the costs if all commercial users of the free OSS had to replace that software with a pecuniary alternative (similar to the calculations Greenstein and Nagle (2014) and Murciano-Goroff et al. (2021) performed for web servers). We can then use this alternate value of p, combined with the q values from above to estimate a goods replacement value of OSS. This method builds on suggestions from Nordhaus (2006, p. 146) who says "…the price of market and nonmarket goods and services should be imputed on the basis of the comparable market goods and services." We do not expect the estimates from the two methods to be similar. Quite the contrary, the value of those two methods varies substantially since the latter goods-market approach assumes a fixed price to sell a good multiple times and that fixed price is usually lower than the total value estimated from recreating all packages on the labor side. The differential between these estimates is essentially the result of a firm stepping in to reproduce the missing OSS packages and then selling them for a pecuniary price rather than all firms needing to reproduce those packages from scratch themselves.

With the goods market approach, the thought experiment is still that we live in a world where OSS does not exist, but it has to be recreated via one firm that then charges a price for a good that is currently free. To value OSS via the goods market approach, we created a basket of equivalent substitute proprietary goods that are priced on the open market as a stand-in for an OSS product. This methodology is consistent with that used in the prior literature (Greenstein and Nagle, 2014; Murciano-Goroff et al., 2021) although both of those studies only used a single good rather than a basket since they were focused on only one type of OSS (web servers). Since there is no readily available database for proprietary equivalents of OSS, we conduct a search based on subjective perception of popularity of OSS and we then search for pecuniary, closed-source substitutes for them. The resulting basket of 20 OSS packages with proprietary substitutes is a good representation of the diversity of OSS. The software ranges from media and design software to statistical analysis programs, to database management and web server software.[21]

[21] Table A5 shows the basket of OSS and their proprietary equivalents that we used. To create this basket, we looked for proprietary software that had an OSS equivalent that was similar in its overall function and feature set, and sought to identify pairs of software that, in aggregate, captured a broad and representative set of the types of OSS that exist.

Based on our OSS equivalent substitute proprietary goods basket we then obtain prices for each proprietary software equivalent and we calculated the COCOMO labor market supply-side value for each OSS product in the basket which we use as a proxy – for lack of the code from the proprietary software – for the COCOMO labor supply side value of the proprietary software (e.g., the cost it would take to pay a programmer to write that proprietary software from scratch). We then calculate the average COCOMO labor supply-side value of the basket, $V_{S,Basket}$, and the average price for the basket, $P_{S,Basket}$.[22] Since, we know the labor market supply-side value of all OSS, V_S^{Labor}, we can setup the following equation and obtain the price of OSS, P_S for the goods-market for all OSS via a simple scaling transformation:

$$\frac{P_S}{V_S^{Labor}} = \frac{P_{S,Basket}}{V_{S,Basket}}$$

$$P_S = \frac{V_S^{Labor}}{V_{S,Basket}} P_{S,Basket} \quad (4)$$

This results in the following goods-market supply-side value:

$$V_S^{Goods} = P_S * 1. \quad (5)$$

We can then obtain the equivalent demand-side goods-market values as follows:

$$V_D^{Goods} = P_S * Q. \quad (6)$$

From the goods market side, we consider the equivalent price that a firm would charge if it produced all existing widely used OSS and then sold it as a product to customers.

Table A6, column 3 shows that the value on the demand side ranges between $177 million and $244 million across the different buckets independent of the country of origin from which programmers are hired. Naturally, the goods-market value will be substantially smaller than the labor demand value since this imaginary firm will make a profit at a comparatively low price by producing the software once and then selling it to many customers. Further, while Greenstein and Nagle (2014) as well as Murciano-Goroff et al. (2021) focus on one particular type of software only (web servers), we attempt to expand on their approach to many goods. However, the data limitations become a stronger constraint in this context which makes it inherently more difficult

[22] We obtained the average price of the goods-market proprietary basket by using a 3 years lifespan of software, i.e. 1/(1-0.66), so the depreciation factor is 0.33. This is consistent with the United States Internal Revenue Service (IRS) rules for depreciating software, which states "If you can depreciate the cost of computer software, use the straight line method over a useful life of 36 months." https://www.irs.gov/publications/p946#en_US_2022_publink1000107354.

to estimate the value through this approach while requiring substantially more assumptions. Instead of a pure goods-market approach, this requires assistance from the labor-market approach in terms of scaling for the estimate, which ultimately leads to a substantial underestimate of the value of OSS. Further, the pricing strategy of the proprietary software counterparts is sensitive to the market demand. With the goods-market approach extending to multiple goods, the strong implicit assumption is that the market demands for our basket software and our sample OSS are similar, so they lead to commensurate prices.

An alternative and even more simplified goods-market back-of-the-envelope calculation that does not account for lines of code and relies on no scaling from the labor market approach would simply be to multiply a price of a reference good with the usage. We can take the minimum, average, and maximum prices of the basket of proprietary goods as captured in the reference price vector p = (69.99, 1610.17, 5800) and simply multiply it with usage from the combined Census and BuiltWith sample (Table 1). Based on those imputed proprietary price assumptions, the goods market demand side value would range from $0.2 – $16.5 trillion (minimum price – maximum price) with the mean price resulting in a value estimate of $4.5 trillion. However, while this estimate creates slightly more variance than the labor market approach, it is inherently flawed by simply assuming that the imputed prices are identical for each open source product based on the reference price. We further think this very simple back-of-the-envelope is an orange to apples comparison since the goods in our basket are fully functional, stand-alone software packages, while the packages in the Census and BuiltWith datasets are comprised of application libraries, which are generally smaller than such stand-alone packages.

Given all the complexities and assumptions one has to make due to lack of data for a goods-market approach, we place stronger emphasis on the labor cost approach highlighted in the main body but we include this method here for completeness.