Practices in the open-source machine learning community: examining the sociotechnical in model documentation on Hugging Face

Word count: 4979

Candidate number: 1087056

Table of Contents

Introduction

In May 2023, a leaked memo written by a Google researcher appeared on Discord, sending ripples around the artificial intelligence (AI) and machine learning (ML) landscape. Its author claimed that while Google had focused on competition with OpenAI the "uncomfortable truth" was that neither Google nor OpenAI was positioned to win the AI "arms race". "While we've been squabbling," they wrote, "a third faction has been quietly eating our lunch" (Patel & Ahmad, 2023): the open source machine learning community.

Alongside the headline catching research and development conducted by OpenAI, Google, Meta, Anthropic, and Mistral, a large amount of parallel activity is taking place in the open source machine learning community. Individuals and institutions are training and experimenting, mostly with smaller, more specialised models developed for specific use-cases, publishing models online to publicise them but also to allow others to build on them.

There is speculation that a healthy open-source environment may contribute to reducing market concentration at the foundation model level (where major players have essentially insurmountable advantages) from cascading vertically into markets for specific model implementations (Kapoor et al., 2024). All ML models, however, are prone to identifying and learning biases which they then incorporate into their outputs, whether that be through subtle and overt biases in text and image generation (Bender et al., 2021; Bianchi et al., 2023; Mehrabi et al., 2022) or in making harmful predictions with discriminatory outcomes (as in the infamous COMPAS case) (Angwin et al., 2016).

This paper examines open source machine learning model documentation practices, focusing its analysis on Hugging Face, the dominant hub for open source ML publication and collaboration which counted 570,282 models in its hub as of data collected on 24 April 2024. Recognising the increasing use of trained ML models to perform high-impact tasks in sensitive areas ranging from law enforcement, to medicine, to employment, Mitchell et al. (2019) developed model cards as a framework to encourage transparent model reporting through accessible documentation detailing performance characteristics. The intention is that this will minimize inappropriate model implementation, use, and re-use. It is thus crucially important to evaluate the effectiveness of the voluntary documentation-centred approach to fair, transparent, and responsible machine learning practices.

Research question:

To what extent have model cards been adopted as a means of documenting the sociotechnical risks and limitations of open source machine learning models on Hugging Face?

Literature review

Open source contextualised

The histories and foundations of both the Internet and artificial intelligence are heavily influenced by open source. Software development's early decades were run on a culture of open access and free exchange between institutional engineers, amateurs, and academics. TCP/IP, DNS, HTTP, SMTP, Thunderbird (IMAP), OpenSSL (SSL/TLS), HTML, CSS, and JavaScript are but a few of the most important open standards and open-source software projects forming the backbone of the modern Internet and our contemporary networked existences.

But open source is more than protocols. The free software movement emerged in the 1970s as a response to the proprietary mechanisms, notably copyright, that began to permeate the tech industry as it started to demonstrate profitability (or at least promises of profitability) (Stallman & Williams, 2010). Richard Stallman, widely considered to be its founder, founded the Free Software Foundation in 1985. The open-source software movement splintered from the wider free software movement in the 1990s, partly in an attempt to develop a more corporate-friendly version of this philosophy (Corrado et al., 2018).

Between the 1990s and early 2000s, something of an open-source boom could be observed, particularly in terms of academic interest in the phenomenon (e.g. Benkler, 2002; Lerner & Tirole, 2001; Oreg & Nov, 2008; Raymond, 2001; Weber, 2005). This initially fed optimistic techno-imaginaries about the Internet and the technologies and platforms that could be built around it as egalitarian, and democratising (Bory, 2020). However, at least in terms of consumer technologies, open source examples such as Wikipedia and Mozilla Firefox represent exceptions to the general rule.

Open source, while remaining hugely influential has tended to exist outside of the spotlight, often providing most of the foundational back-end architecture of products while companies focus their intellectual property defence of the user-facing layers of stacks (Ackermann, 2023). The vulnerability in Apache Log4J, a Java-based open-source logging utility served as a powerful reminder of this when it put billions of devices at risk of exploitation in 2021 (O'Neill, 2021). In early 2024, the revelation that somebody had intentionally placed a backdoor in XZ Utils, an open source data compression utility available on almost all Unix-like operating systems, once again brought to attention the widespread influence of open source software. In its 2024 Open Source Security and Risk Analysis Report, US cybersecurity firm Synopsys found that 96% of commercial codebases it analysed contained open source code. And this was not just part of their code: 77% of all code in these codebases had open source origins (Synopsys, 2024).

This matters because it illustrates how important and prevalent open source software already is. There are strong reasons to believe that many future products and technologies incorporating machine learning will include at least some elements originating from open source projects. This means the risks, flaws, system vulnerabilities, and unresolved fairness and safety issues of these can be inherited and left unaddressed if the downstream user is not made aware of them.

Open source machine learning

Open source machine learning research and development has been flourishing in recent years, particularly since Meta's LLaMA model was leaked (and then released) in mid-2023. As Corrado et al. (2018) note, open source has its origins not just in the Free Software Movement but in the cultures of sharing and collaboration that were fostered hundreds of years earlier by the Royal Society which recognised the importance of openness for scientists to both systematically reproduce experiments and results and to build on each other's findings. ML research being until recently a predominantly scientific, rather than industrial, field, many important advances within it have been shared publicly in an open source manner. It is the recent move towards commercialisation, combined with the rapid progress in model capabilities, that has created unique open versus closed tension in the field.

Critics of open source ML models (e.g. Harris, 2024) argue that the rapid and uncontrolled release of powerful and unmoderated models poses risks to information ecosystems, and therefore our democracies, by empowering people and groups to engage in the widespread distribution of personalised misinformation. Alongside this, they highlight the risks posed by increasingly sophisticated scams, damaging non-consensual deepfake pornography, cyberattacks, and the possibility for open source models to facilitate the production of biological and chemical weapons. While model cards cannot address the malicious use issue (Seger et al., 2023), a recent study conducted by Stanford's Center for Research on Foundation Models has found that the marginal risk of open source foundation models (the extent to which these models increase societal risk by intentional misuse beyond closed models or pre-existing technologies) is currently for the most part low (Kapoor et al., 2024).

However, unintentional misuse or misapplications of models are likely to far outnumber examples of malicious use as they are implemented worldwide across industries and the public sector. Well-intentioned commercial, scientific, and personal applications based on freely available open source ML models will propagate the risks, flaws, system vulnerabilities, and unresolved safety issues of the models they are based on (Seger et al., 2023). Documentation such as model cards are crucial to reducing the risk of this occurring by including sociotechnical information which raises downstream users' awareness about the risks and weaknesses of the models they are building with. The intention is that they either choose not to use inappropriate or problematic models, or know they need to address the issues they inherit before implementing them into their own use cases. Model cards are a key means of achieving Kapoor et al. (2024)'s recommendation that developers of open models be transparent "about both the responsible AI practices they implement and the responsible AI practices they recommend or delegate to downstream developers or deployers".

Model cards

Model cards were proposed by Mitchell et al. (2019) as a standardized documentation procedure "to communicate the performance characteristics" of trained machine learning (ML) and artificial intelligence (AI) models. Though they were designed to include technical information (training data, datasets), Focusing especially on biased and discriminatory outcomes, they suggested these should provide benchmarked evaluation in a variety of

conditions including across cultural, demographic, or phenotypic groups as well as intersectional groups relevant to the intended application domain. Alongside these, they suggest model cards should disclose the context in which models are intended to be used, as well as performance evaluation procedures and other relevant information.

They suggest model cards should contain nine sections: Model Details, Intended Use (and Out-of-scope uses), Factors, Metrics, Evaluation Data, Training Data, Quantitative Analyses, Ethical Considerations, Caveats, and Recommendations (see Appendix 1A). Though by no means a criticism, it is important to recognise that this represents a significant undertaking on the model developer's behalf, particularly when this practice is entirely voluntary. The 'Factors' section, for instance, suggests developers should provide a summary of model performance across groups (e.g. people who share one or multiple characteristics), instrumentation (e.g. across different cameras for facial recognition models), and environments (e.g. lighting and humidity for facial recognition models). The 'Quantitative Analysis' section, meanwhile, involves extensive disaggregated quantitative analyses to be conducted and presented, demonstrating performance differences across different metrics (e.g. race, age, gender, and intersectional combinations) (Mitchell et al., 2019).

Though Hugging Face strongly emphasises the importance of including model cards for models uploaded to the platform, they are not mandatory (Hugging Face, n.d.). Research conducted in 2022 found the sections most relevant to fairness and responsible AI/ML development and most highlighted by Mitchell to have the lowest filled-out rates (Liang et al., 2024). This paper will seek to complement Liang et al.'s research by considering the landscape two years after their data was collected and by examining whether and how trends have changed over time.

Methodology

ML model documentation being very much a nascent field of study, very few studies have been conducted examining their uses and contents. Empirical studies have been conducted using manual analysis (Pepe et al., 2024), but computational approaches are scarce. Shortly after beginning to study Hugging Face model cards, Liang et al. (2024) published their systematic analysis of ML model cards on Hugging Face using a sentence-level topic modelling approach to identify patterns and themes within the text. Lacking the time and

means to do this and having 169,556 model cards to analyse as opposed to the 32,111 they collected in October 2022, analysis of the model cards' content will primarily be conducted using a keyword search method analysed alongside mean wordcount data and metadata including creation dates and downloads.

Data collection

After initially trialling an approach using web scraping, the Hugging Face Hub API, huggingface_hub, was used to access data on all models uploaded to the Hugging Face Hub including their Model Cards which are attached as README files written in markdown format.

A master list of all models uploaded publicly to Hugging Face was collected using HfApi.list_models(). On 24 March 2024, when the data used for analysis was collected, this list was 570,283 models long. Using Python, a function was written to load the model cards associated with each model from the master list's id using HfApi.ModelCards.load() function. Errors were handled in such a way that any error reading 404, which meant a model card was not uploaded for the specific model, would output a blank model card. Concurrent execution using ThreadPoolExecutor() enables parallel processing in batches of 1000, optimizing efficiency. Separately, metadata on each model's author, publication date, number of downloads, number of likes, and tags with information including tasks (multimodal, computer vision, natural language processing, audio, etc.), libraries, datasets, languages, and licences. The processed data was converted into a pandas DataFrame, then into an Arrow table, and saved as a Parquet file for further analysis. Parquet was used to save the data because of the size of the dataset (570,000 rows) since it organizes data in a columnar fashion, allowing for better compression and encoding, leading to reduced storage space and improved query performance. See Appendix 3A for sample code.

The mistune library (https://mistune.lepture.com/) was used to parse the downloaded model cards' README files, removing formatting elements, tables, images, urls, and markdown front matter containing metadata and configuration information. This returned plain .txt files with only the core text content and headings of the model cards retained. For this code, see Appendix 3B.

Findings

Model card presence

As of 24 March 2024, when the data for this research was collected, 570,283 models had been uploaded to Hugging Face. Of these, 169,671 (29.8%) were uploaded with model cards, of which 169,556 (29.7%) contained at least one word. The mean number of words across all model cards was 50.1, rising to 168.5 if empty model cards were discounted. Though the number of models published on Hugging Face is growing at an exponential rate (Figure 1), this is primarily pushed by growth in the publication rate of models with no or empty model cards. The aberration observable in March 2022 can be explained by the fact that the publishing date for all models published before that date is set to '2022-03-02'.

Figure 1a.

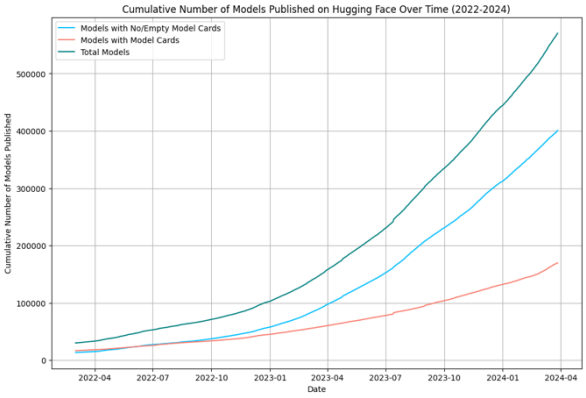
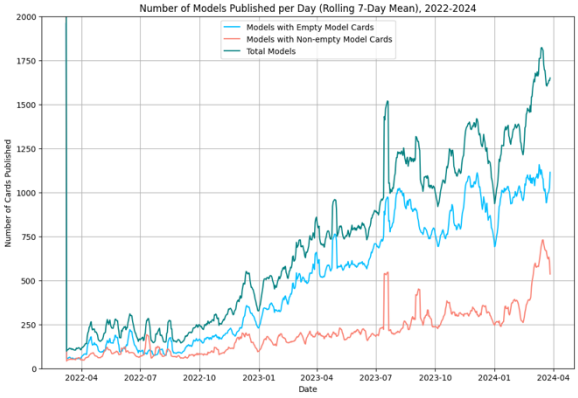


Figure 1b.



As one might expect, models without a model card tend to have fewer downloads and make up the vast majority of the models with no downloads (rank 269,595 onwards). This explains the sharp drop observed in Figure 5a at that mark. That said, 14.04% of the top 10,000 models (the lowest ranked of which has 1291 downloads) nevertheless have no model card and thus no technical or sociotechnical information.

Model card length

Among the models with model cards, a number of very large outliers with extremely high wordcounts were identifiable. 20 model cards had over 5,000 words, most of them topic models with very large dropdown tables providing an overview of all topics (e.g. KingKazma/cnn_dailymail_108_50000_25000_train) or topic visualisations (e.g. MaartenGr/BERTopic_Wikipedia). Applying the IQR method of identifying outliers to the subset of models that had model cards resulted in 143,587 models and a mean of 83.1 words. However, excluding all models above 258 words, as this does due to the wide spread of the data, is inappropriate. Instead excluding the top percentile or half-percentile of models with model cards by word count leaves 167,859 or 168,707 models and a mean word count of 147.2 or 155.56 respectively. Manual verification of the top remaining models by word count after exclusion of the top percentile suggests this is appropriate as their high word counts are not the consequence of the presence of large, generated topic tables.

Table 1: Word count statistics

	All	**Non-empty**	**IQR-outliers removed**	**Top 1% outliers removed**	**Top 0.5% outliers removed**
# model cards	570,283	169,556	143,587	167,859	168,707
Mean	50.10	168.50	83.14	147.18	155.46
Median	0.0	88.0	86.0	88.0	88.0
Range	103,980	103,980	257	1671.0	1927.0
Mode	0	86.0	86.0	86.0	86.0
Std	237.01	411.07	49.60	207.88	237.89

While it is hard to draw many conclusions from these statistics given the spread of the data and number of outliers (see the standard deviation statistics in Table 1), once the most significant outliers are removed, the mean word count of model cards can conclusively be said to be low (Table 1). For reference, the example model cards provided by Mitchell et al. (2019) were 414 and 309 words long. Examining the distribution of the word count data, this impression is confirmed and compounded. As the histograms in Figure 2 illustrate, the length of model cards is heavily right-skewed and is particularly concentrated around the 80-100 word count. This is significantly lower than anything that can contain the depth of

information envisaged by Mitchell et al. to cover the sociotechnical aspects relevant to the models published.

Figure 2a.

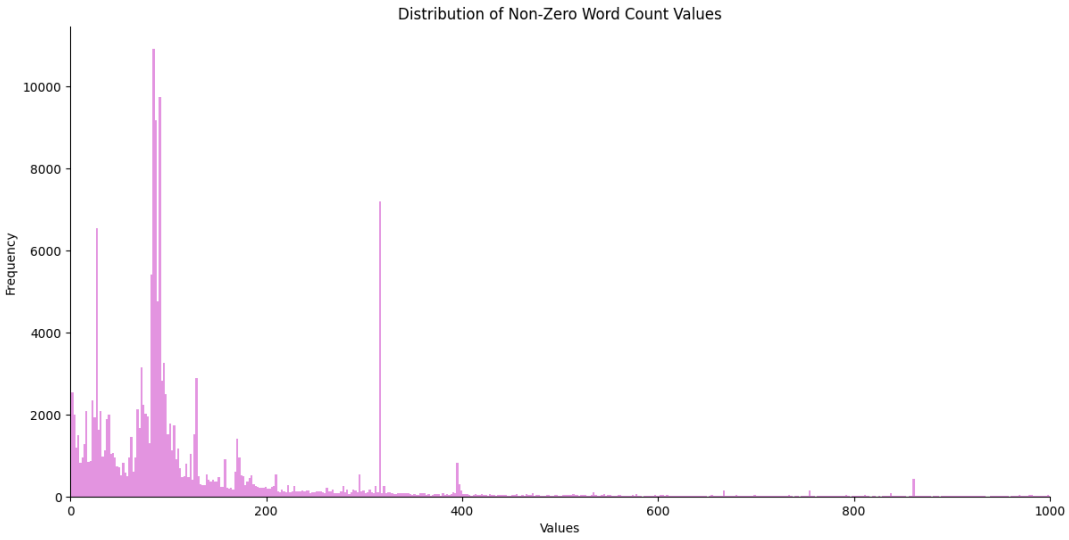
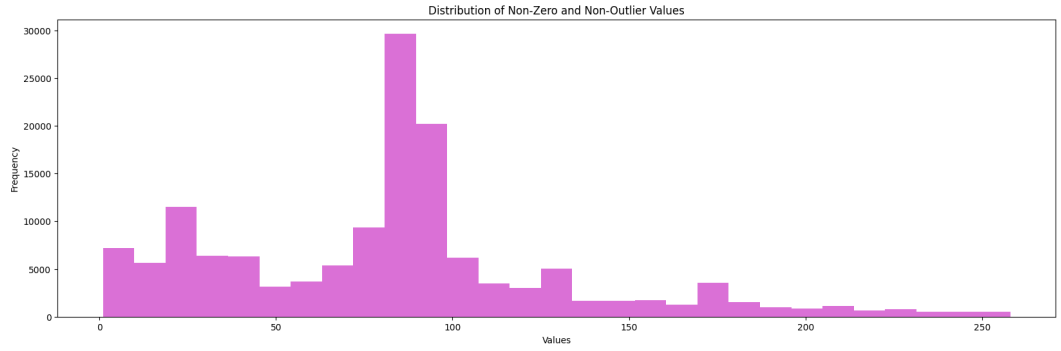


Figure 2b.



While the volume and spread of the data collected makes it challenging to analyse by looking at individual data points, grouping the models by downloads and comparing those yields interesting trends. In particular, we find a reasonably strong correlation between downloads and averaged word counts (Figure 3a). This general trend is apparent when we examine means across models broken into groups of 10,000 models, but also when we perform the same analysis with models analysed in groups of 1,000 (Figure 3b). This suggests the trend is relatively strong, confirmed by polynomial regression analysis of the 10,000 model chunks (Figure 3a). While this does not mean a causal relationship can be identified, it does suggest that there is a strong association between model card length and downloads.

Interestingly, observing Figure 3a, we find that there is an abrupt cut-off point after the 270,000th model when ranked by downloads. Almost all groups of 10,000 thereafter have an average word count of ~0, with only ranks 460,000-500,000 exceeding ~0, and none of them with an average of above 10 words. This suggests that while a majority of the models uploaded to Hugging Face have no model card, and many of them have model cards with word counts too low to include sociotechnical information, the models that are downloaded the most have longer-than-average model cards and are therefore the most likely to also include sociotechnical information. When we plot the average number of downloads of each of these 10,000-model chunks alongside the mean word count (Figure 3a), we see that this drops to zero at around the 270,000 model mark as well, lending further credence to the hypothesis that there is a direct relationship between model card presence and length, and downloads. However, while this lends validity to the importance of model card length, it tells us nothing about the importance of different types of content.

Figure 3a.

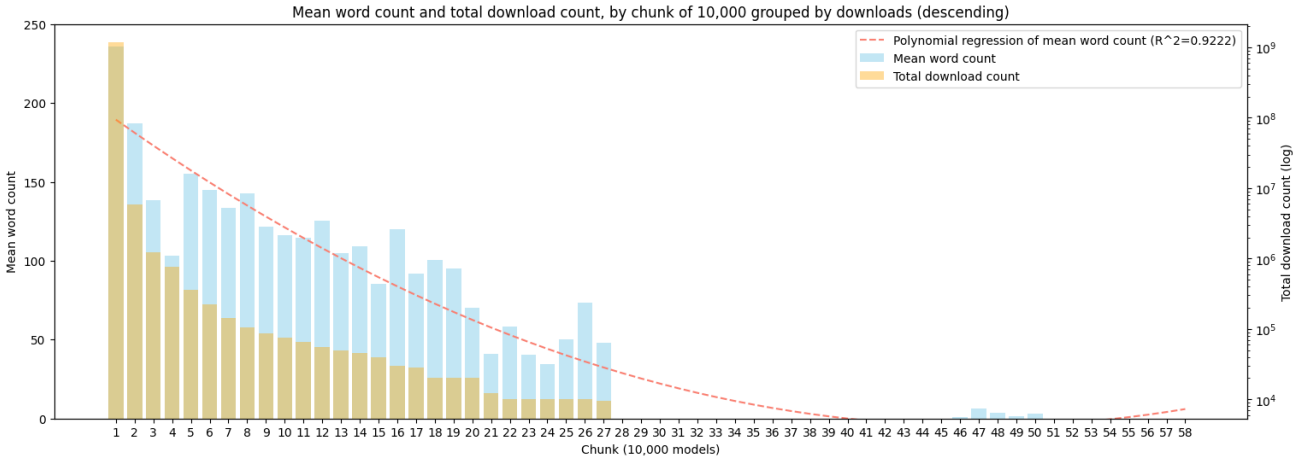


Figure 3b.

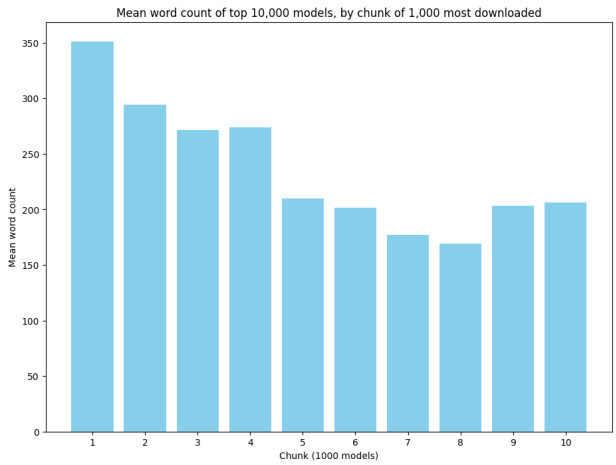
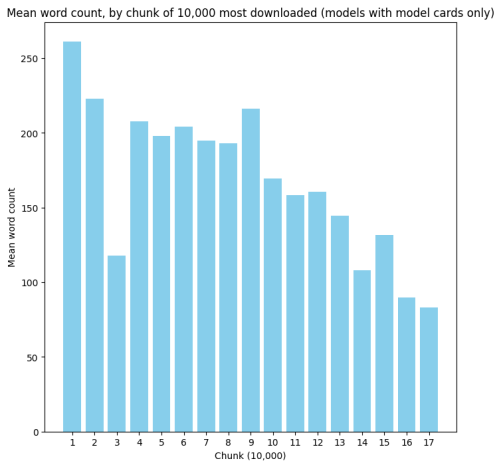


Figure 3c.



Turning to an analysis of mean word count trends over time, we find little suggestion that model card word counts are increasing over time, or at least not in a consistent or clear manner (Figure 4a). This is in line with the finding that more and more models are being published with no model card. For reasons that are unclear, September 2023 experienced a particularly high average word count. The months after September 2023 (up to March 2024) also seem to exhibit an upward trend in the monthly mean word count. Discounting models with no cards and examining only the mean word count of models with model cards in that same period (Figure 4b), we observe relatively consistent growth across the 2023 calendar year before a drop across the first three months of 2024. Various outlier-removal methods have also been shown on Figure 4b to demonstrate the uncertainty of these suggested trends and the role extremely high word count outliers might play in driving them (as opposed to a more general increase in model card length across all models). It is thus challenging to draw any substantial conclusions from an analysis of mean word count across time.

Figure 4a.

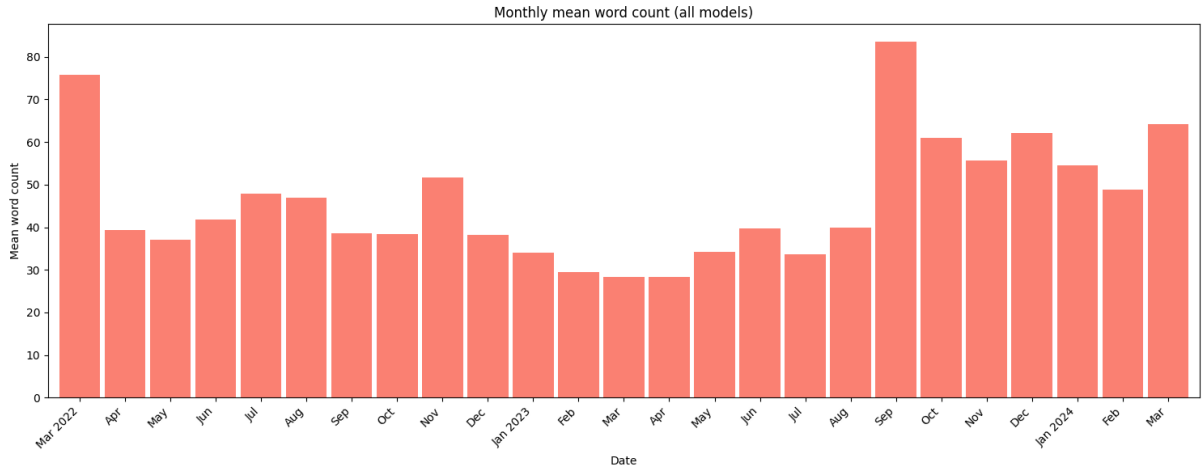
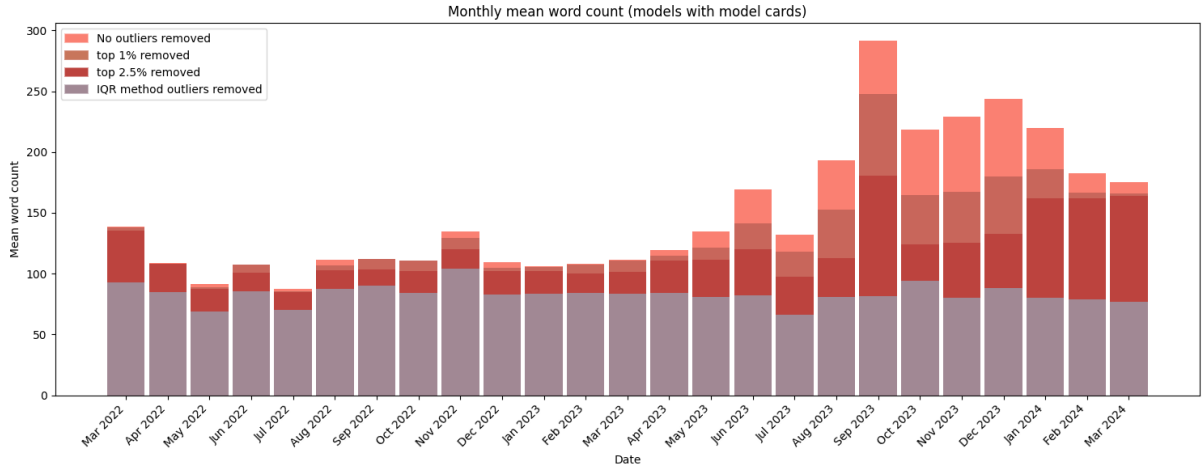


Figure 4b.



Model card content

Moving to an analysis of the content of the model cards, the presence of keywords, headings, and phrases was examined to gain insights into what publishers are including in their model cards if and when they do create them.

Heading analysis

The model card template provided by Hugging Face on their website as a markdown file (Appendix 2) serves as a starting point as it was developed following Mitchell et al. (2019)'s work. Two key headings are used for sociotechnical sections: 'Bias, Risks, and Limitations', and 'Out-of-Scope Use'.

While 9,638 model cards (5.68% of model cards; 1.69% of models) included the heading 'Bias, Risks, and Limitations' suggested by the template, 8,312 (86.26%) of these were followed by '\n\n[More Information Needed]'. This means only 1,324 models (0.78% of model cards; 0.23% of models) used the template (or part of it) and added information to the 'Bias, Risks, and Limitations' section. 'Out-of-Scope Use' was used as a heading in 9,376 model cards (5.53% of model cards; 1.64% of models), but it was followed by '\n\n[More Information Needed]' in 8,307 cases (88.65%). Only 1,069 (0.63% of model cards; 0.19% of models) used the template (or part of it) and filled in the 'Out-of-Scope Use' section.

Given that most models (approximately 94%) did not use the template (or at least did not use the proposed Bias, Risks, and Limitations and Out-of-Scope Use sections of them), further investigation is necessary to ascertain the extent to which model cards are filled, especially with information related to foreseeable harms, misunderstandings, and technical and sociotechnical limitations.

Keyword analysis

A list of 49 sociotechnical keywords was compiled based on Mitchell et al. (2019) paper, the literature on sociotechnical risks associated with machine learning cited above, and a qualitative analysis of a sample of the most downloaded models' model cards.

The keywords searched for (Appendix C) appear in 86,307 of the models (50.90% of model cards; 15.13% of models). However, as soon as ‘limitation’ and ‘limitations’ are removed from that search, that number drops significantly to 22,204 (13.10% of model cards; 3.89% of models). We can also remove any model including either ‘Bias, Risks, and Limitations\n\n[More Information Needed]’ or ‘Out-of-Scope Use\n\n[More Information Needed]’ as these are strongly presumed to have used the Hugging Face model card template and left both of those key sections empty. We considering them to include the keywords but in fact be devoid of any content of sociotechnical value. We thus find that 13,855 of the models (8.18% of model cards; 2.43% of models) include any of the selected sociotechnical keywords – again an extremely low number.

An nltk analysis of the most common words used in the model cards using a counter (Appendix 2b and 3e) ranks ‘bias’ as the top sociotechnical keyword with 23,479 occurrences (rank 78). Technical words (e.g. ‘hyperparameters’, ‘transformers’, ‘procedure’) dominate the rankings, suggesting model cards are dominantly used to share technical, rather than sociotechnical, information.

Taking these 13,855 models and merging them with the list of models organised by downloads and broken into groups of 10,000, we can examine the relationship between downloads and the inclusion of sociotechnical keywords in model cards. While 15.74% of the top 10,000 models include at least one keyword, this drops below 10% for the models ranked 20,000 to 70,000 but is again over 10% for ranks 70,000-100,000 (peaking at 17.55% for the models ranked 70,000-80,000). Though the global trend is for higher proportions of more downloaded models to include keywords, the correlation is far from strong. The presence of sociotechnical keywords is thus highly unlikely to influence download numbers to any significant extent.

Turning to an analysis over time, discounting March 2022, we find (Figure 5b.) no noticeable sustained change in the percentage of model cards (let alone models) published with sociotechnical keywords. While the a very slight upward trend can be observed between January 2023 and January 2024, the averaged rates for that period are lower (6.55%) than they were for April 2022-November 2022 rates (8.28%).

Figure 5a.

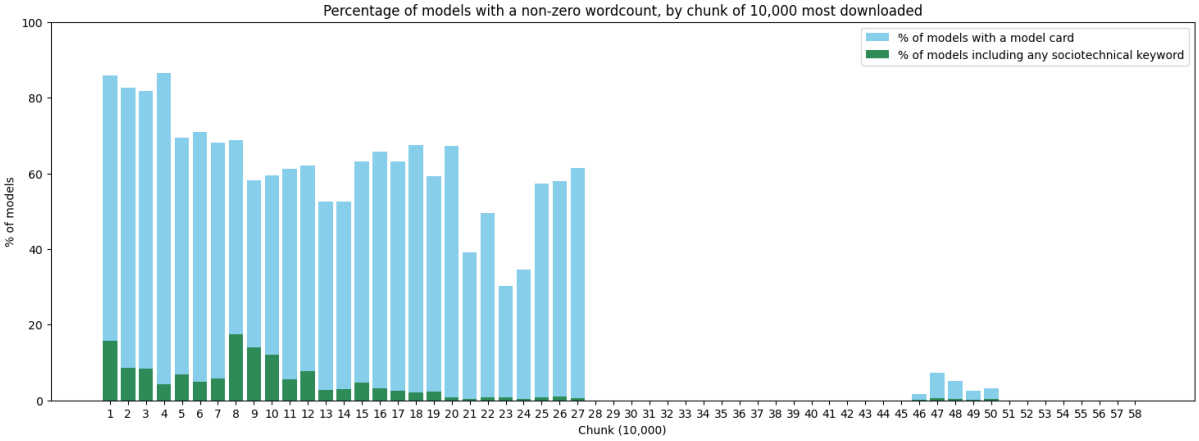


Figure 5b.

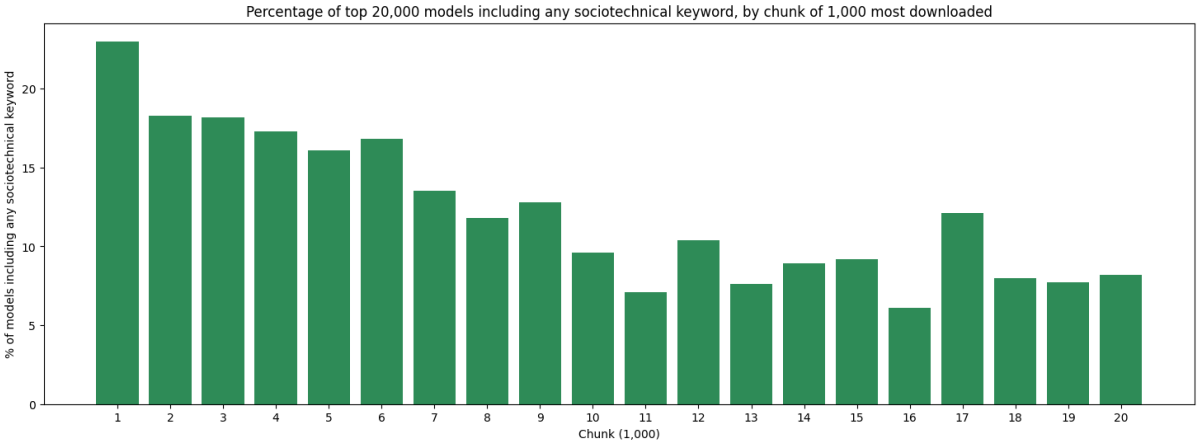
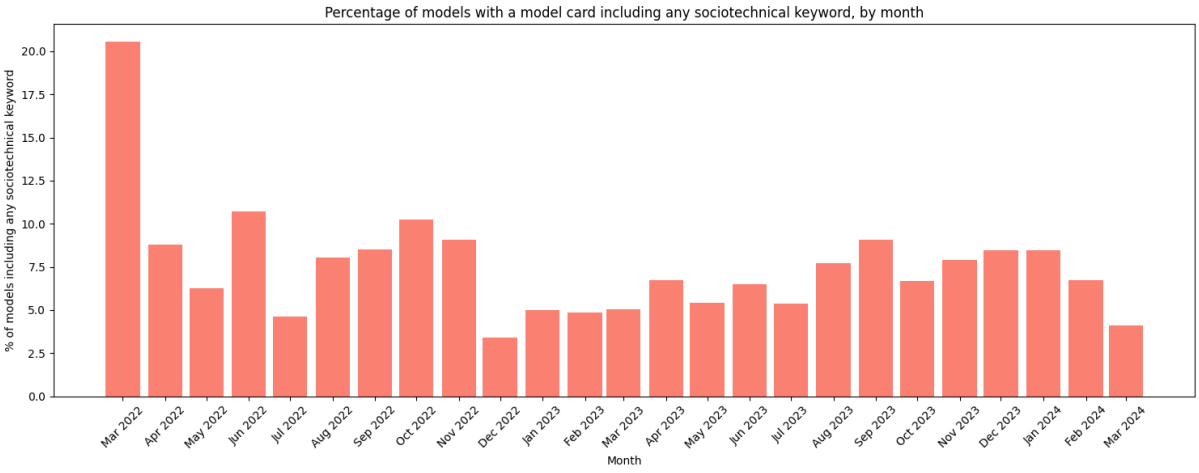


Figure 5c.



Conclusion

The intention of this paper is to examine model documentation practices in open source machine learning and artificial intelligence. Focusing on Hugging Face, the dominant hub for open source ML model publication and collaboration, 570,283 models were analysed for their use of model cards. The research focused on the extent to which they cover the sociotechnical elements highlighted as central to a model card by Mitchell et al. (2019) when they developed the conceptual framework for the model card concept Hugging Face bases its documentation recommendations on. Model card presence, length, and content were analysed with respect to download numbers and publication date to examine possible correlations and trends. This resulted in five main findings.

First, only 169,556 (29.7%) of the model repositories include model cards uploaded as a README file and visible on the front page of the repository's page on the Hugging Face Hub. This is a remarkable decrease compared to 2022, when 44.2% of repositories included model cards (Liang et al., 2024). Analysis of how this has changed over time (Fig 1a) suggests the ratio of model card free models is rising fast and shows little sign of slowing down.

Even among those models that do provide model cards, we find that the mean length of these is extremely short (155.46 words when excluding the top 1% of outliers), especially when compared with the sample model cards provided by Mitchell et al. (2019) which were 414 and 309 words long. The right-skewedness of this data and the remarkably high density of frequencies around the 80-to-100-word length confirms that a vast majority of model cards that are published are almost certainly too short to contain any useful sociotechnical information.

While a relatively strong positive correlation was identified between model card word count and total downloads, no such general relationship was found to exist between downloads and the presence of sociotechnical information. However, when confining the analysis of this second relationship to the top 20,000 models in terms of downloads (and especially the first 10,000), a positive correlation is identifiable. The very most downloaded models thus have moderately higher ratios of sociotechnical content. Though this remains low in the wider scheme of things (23.0% for the top 1000; 28% for the top 100), given that download counts are extremely concentrated among the highest ranking models (Figure 3a), it .

Overall analysis of the content of the published model cards using keyword search of 47 relevant sociotechnical keywords found only 13,855 of the models (8.18% of models with model cards; 2.43% of models) contained any one of those keywords. This is extraordinarily low, and although different methods were used to their research, implies a decline in the presence of sociotechnical content in model cards compared to Liang et al.'s 2022 findings. The nltk count which revealed the dominance of technical vocabulary aligns with their previous findings.

Finally, a historical trend analysis between March 2022 and April 2024 does not suggest any particular improvement in model card length, nor in the ratio of models including sociotechnical information. If anything, the second of these has slightly decreased (to around 6.5%) since the end of 2022.

The question posed at the beginning of this paper –to what extent have model cards been adopted as a means of documenting the sociotechnical risks and limitations of open source machine learning models on Hugging Face– can be answered with an emphatic answer: very little. While public discourse about fairness, accountability and transparency in AI and ML is becoming more prominent and widespread as concerns about risks of societal harm caused by advanced models rise, this is not reflected in the practices of model publishers in the open source machine learning community.

The exact role open source will play in future widespread implementations of ML models remains uncertain, and we cannot yet predict its influence. However, lessons from the wider history of open source software suggests the possibility of both being significant is not at all unlikely. This is particularly true as foundation models such as Meta's upcoming LLaMA 3 continue to be made publicly (or partially publicly) available. Because one of open source's core raisons d'être is downstream use, either as part of an adjusted or improved model, or within a larger system, including information about both technical and sociotechnical caveats in open source models for future implementors but also those auditing them, is critically important. When the open source software in question is ML models, with the widely documented sociotechnical risks that accompany many of them, this is only compounded (Bender et al., 2021; Bianchi et al., 2023).

This paper's core findings suggest that the model card documentation solution proposed by Mitchell et al. (2019) is failing to gain any meaningful voluntary adoption among the open

source community of ML developers. Model cards are for the most part being used to present exclusively technical information about their associated repositories. One of the much-heralded benefits of open source is in how it democratises software and development, allowing hobbyists to contribute alongside immense corporations. This complicates the question of how to best ensure the potentially problematic and harmful aspects of publicly and freely available models are not left unaddressed due to ignorance as to their presence. How to go about this is a question requiring further thought and research, but this paper has conclusively demonstrated that voluntarily completed model cards are not, and show no sign of becoming, the solution to these sociotechnical challenges.

Limitations and further research

While this paper provides a number of insights into model card practices on Hugging Face, its use of keyword search for content analysis certainly means a number of nuances are lost. Though more fine-grained analysis could be conducted using keywords, applying topic modeling approach used by Liang et al. to the data collected for this paper likely yields greater insights.

Qualitative research also has an important role to play, particularly in explaining some of the findings in this paper. Questions raised such as why so many models are published with no model cards and receiving on downloads, while a small number of models are downloaded millions of times, might be explored. Interaction with players in the open source community through interviews or ethnographic research might also assist in providing better explanations as to why model card practices are as this paper has found them to be, notably why they are deficient in sociotechnical content.

Finally, though HuggingFace is the dominant platform for open source machine learning model sharing, it is not the only one. GitHub in particular continues to be used as an alternative location for sharing models. So while a focus on practices on HuggingFace certainly gives important insights and is highly likely to be broadly representative of trends in general open source machine learning practices, research intro trends on other platforms is also needed.

Bibliography

Ackermann, R. (2023, August 17). *The future of open source is still very much in flux*. MIT Technology Review. https://www.technologyreview.com/2023/08/17/1077498/future-open-source/

Angwin, J., Larson, J., Mattu, S., & Kirchner, L. (2016, May 23). *Machine Bias*. ProPublica. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623. https://doi.org/10.1145/3442188.3445922

Benkler, Y. (2002). Coase's Penguin, or, Linux and 'The Nature of the Firm'. *The Yale Law Journal*, *112*(3), 369. https://doi.org/10.2307/1562247

Bianchi, F., Kalluri, P., Durmus, E., Ladhak, F., Cheng, M., Nozza, D., Hashimoto, T., Jurafsky, D., Zou, J., & Caliskan, A. (2023). Easily Accessible Text-to-Image Generation Amplifies Demographic Stereotypes at Large Scale. *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, 1493–1504. https://doi.org/10.1145/3593013.3594095

Bory, P. (2020). *The Internet Myth: From the Internet Imaginary to Network Ideologies*. University of Westminster Press. https://uwestminsterpress.co.uk/site/books/m/10.16997/book48/

Corrado, E. M., Sandy, H. M., & Mitchell, E. T. (2018). Nullis in Verba: The Free Software Movement as a model for Openness and Transparency. *Technical Services Quarterly*, *35*(3), 269–279. https://doi.org/10.1080/07317131.2018.1456849

Harris, D. E. (2024, January 12). *Open-Source AI Is Uniquely Dangerous*. IEEE Spectrum. https://spectrum.ieee.org/open-source-ai-2666932122

Hugging Face. (n.d.). *Model Cards*. Hugging Face Hub. Retrieved 2 April 2024, from https://huggingface.co/docs/hub/en/model-cards

Kapoor, S., Bommasani, R., Klyman, K., Longpre, S., Ramaswami, A., Cihon, P., Hopkins, A., Bankston, K., Biderman, S., Bogen, M., Chowdhury, R., Engler, A., Henderson, P., Jernite, Y., Lazar, S., Maffulli, S., Nelson, A., Pineau, J., Skowron, A., … Narayanan, A. (2024). *On the Societal Impact of Open Foundation Models* (arXiv:2403.07918). arXiv. http://arxiv.org/abs/2403.07918

Lerner, J., & Tirole, J. (2001). The open source movement: Key research questions. *European Economic Review*, *45*(4–6), 819–826. https://doi.org/10.1016/S0014-2921(01)00124-6

Liang, W., Rajani, N., Yang, X., Ozoani, E., Wu, E., Chen, Y., Smith, D. S., & Zou, J. (2024). *What's documented in AI? Systematic Analysis of 32K AI Model Cards* (arXiv:2402.05160). arXiv. http://arxiv.org/abs/2402.05160

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2022). *A Survey on Bias and Fairness in Machine Learning* (arXiv:1908.09635). arXiv. http://arxiv.org/abs/1908.09635

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model Cards for Model Reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220–229. https://doi.org/10.1145/3287560.3287596

O'Neill, P. H. (2021, December 17). *The internet runs on free open-source software. Who pays to fix it?* MIT Technology Review. https://www.technologyreview.com/2021/12/17/1042692/log4j-internet-open-source-hacking/

Oreg, S., & Nov, O. (2008). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, *24*(5), 2055–2073. https://doi.org/10.1016/j.chb.2007.09.007

Patel, D., & Ahmad, A. (2023, May 4). *Google 'We Have No Moat, And Neither Does OpenAI'*. https://www.semianalysis.com/p/google-we-have-no-moat-and-neither

Pepe, F., Nardone, V., Mastropaolo, A., Canfora, G., & Bavota, G. (2024). *How do Hugging Face Models Document Datasets, Bias, and Licenses? An Empirical Study*.

Raymond, E. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly.

Seger, E., Dreksler, N., Moulange, R., Dardaman, E., Schuett, J., Wei, K., Winter, C., Arnold, M., hÉigeartaigh, S. Ó., Korinek, A., Anderljung, M., Bucknall, B., Chan, A., Stafford, E., Koessler, L., Ovadya, A., Garfinkel, B., Bluemke, E., Aird, M., … Gupta, A. (2023). *Open-Sourcing Highly Capable Foundation Models: An evaluation of risks, benefits, and alternative methods for pursuing open-source objectives* (arXiv:2311.09227). arXiv. http://arxiv.org/abs/2311.09227

Stallman, R., & Williams, S. (2010). *Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution*. Free Software Foundation.

Synopsys. (2024). *2024 Open Source Security and Risk Analysis Report*. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

Weber, S. (2005). *The success of open source*. Harvard Univ. Press.

Appendices

Appendix 1a: Summary of model card sections and prompts for each suggested by Mitchell et al. (2019)

Model Card

- **Model Details**. Basic information about the model.
  - Person or organization developing model
  - Model date
  - Model version
  - Model type
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features
  - Paper or other resource for more information
  - Citation details
  - License
  - Where to send questions or comments about the model
- **Intended Use**. Use cases that were envisioned during development.
  - Primary intended uses
  - Primary intended users
  - Out-of-scope use cases
- **Factors**. Factors could include demographic or phenotypic groups, environmental conditions, technical attributes, or others listed in Section 4.3.
  - Relevant factors
  - Evaluation factors
- **Metrics**. Metrics should be chosen to reflect potential real-world impacts of the model.
  - Model performance measures
  - Decision thresholds
  - Variation approaches
- **Evaluation Data**. Details on the dataset(s) used for the quantitative analyses in the card.
  - Datasets
  - Motivation
  - Preprocessing
- **Training Data**. May not be possible to provide in practice. When possible, this section should mirror Evaluation Data. If such detail is not possible, minimal allowable information should be provided here, such as details of the distribution over various factors in the training datasets.
- **Quantitative Analyses**
  - Unitary results
  - Intersectional results
- **Ethical Considerations**
- **Caveats and Recommendations**

Appendix 1b: First section of the model card template provided by Hugging Face

https://huggingface.co/docs/hub/model-card-annotated#template

Model Name

Section Overview: Provide the model name and a 1-2 sentence summary of what the model is.

`model_id`

`model_summary`

Table of Contents

Section Overview: Provide this with links to each section, to enable people to easily jump around/use the file in other locations with the preserved TOC/print out the content/etc.

Model Details

Section Overview: This section provides basic information about what the model is, its current status, and where it came from. It should be useful for anyone who wants to reference the model.

Model Description

`model_description`

Provide basic details about the model. This includes the architecture, version, if it was introduced in a paper, if an original implementation is available, and the creators. Any copyright should be attributed here. General information about training procedures, parameters, and important disclaimers can also be mentioned in this section.

- **Developed by:** `developers`

List (and ideally link to) the people who built the model.

- **Funded by:** `funded_by`

List (and ideally link to) the funding sources that financially, computationally, or otherwise supported or enabled this model.

- **Shared by [optional]:** `shared_by`

List (and ideally link to) the people/organization making the model available online.

- **Model type:** `model_type`

You can name the "type" as:

1. Supervision/Learning Method

2. Machine Learning Type

3. Modality

- **Language(s)** [NLP]: `language`

Use this field when the system uses or processes natural (human) language.

- **License:** `license`

Name and link to the license being used.

- **Finetuned From Model [optional]:** `base_model`

If this model has another model as its base, link to that model here.

Appendix 2a: Keywords, by count

Keyword	Count	% of model cards	% of models
limitation	82764	48,81%	14,51%
limitations	82617	48,73%	14,49%
bias	19099	11,26%	3,35%
risk	12216	7,20%	2,14%
risks	11467	6,76%	2,01%
biases	11404	6,73%	2,00%
out-of-scope use	9903	5,84%	1,74%
environmental impact	8882	5,24%	1,56%
carbon emissions	8646	5,10%	1,52%
biased	2113	1,25%	0,37%
harmful	2064	1,22%	0,36%
misuse	859	0,51%	0,15%
race	854	0,50%	0,15%
water	795	0,47%	0,14%
problematic	671	0,40%	0,12%
stereotype	648	0,38%	0,11%
stereotypes	640	0,38%	0,11%
fairness	529	0,31%	0,09%
mislead	404	0,24%	0,07%
misinformation	401	0,24%	0,07%
out-of-scope uses	400	0,24%	0,07%
danger	382	0,23%	0,07%
misleading	378	0,22%	0,07%
discrimination	347	0,20%	0,06%
racist	276	0,16%	0,05%
dangerous	274	0,16%	0,05%
disinformation	236	0,14%	0,04%
discriminatory	216	0,13%	0,04%
abuse	211	0,12%	0,04%
caveat	171	0,10%	0,03%
caveats	115	0,07%	0,02%
cybersecurity	108	0,06%	0,02%
misused	90	0,05%	0,02%
scam	76	0,04%	0,01%
sexual abuse	62	0,04%	0,01%
pornographic	58	0,03%	0,01%
scams	51	0,03%	0,01%
pornography	43	0,03%	0,01%
stereotypical	36	0,02%	0,01%
prejudice	33	0,02%	0,01%
out of scope use	20	0,01%	0,00%
environmental impacts	8	0,00%	0,00%
prejudices	4	0,00%	0,00%
inequality	4	0,00%	0,00%
intersectional	1	0,00%	0,00%
untrue	1	0,00%	0,00%
inequalities	0	0,00%	0,00%
out of scope uses	0	0,00%	0,00%
Total	86307	50,90%	15,13%

Appendix 2b: 100 most common words in model cards

1	model	748285
2	information	539200
3	needed	516343
4	training	443645
5	following	160293
6	use	152483
7	results	151724
8	data	149711
9	evaluation	148373
10	hyperparameters	144551
11	used	127320
12	dataset	125943
13	optional	125675
14	models	121494
15	using	120409
16	limitations	100756
17	transformers	95778
18	version	89794
19	uses	89765
20	description	87928
21	set	82689
22	procedure	80505
23	datasets	76827
24	intended	75800
25	trained	72397
26	versions	71427
27	framework	69781
28	https	67575
29	pytorch	67422
30	loss	64870
31	optimizer	64677
32	tokenizers	62965
33	seed	59154
34	card	58780
35	language	57886
36	adam	57388
37	achieves	56198
38	linear	53470
39	text	53334
40	code	52959
41	details	52306
42	accuracy	51115
43	inference	46561
44	download	45166
45	usage	42310
46	please	38063
47	files	38037
48	gpu	37284
49	none	36241
50	see	34274

51	paper	33968
52	license	33354
53	example	33011
54	get	32705
55	ai	32671
56	agent	32145
57	original	31088
58	library	30912
59	type	30248
60	prompt	28799
61	metrics	28762
62	parameters	28481
63	also	28466
64	support	28387
65	repository	28131
66	like	28088
67	learning	27201
68	format	26892
69	train	26720
70	based	26601
71	FALSE	26517
72	generation	26469
73	python	25710
74	available	25352
75	work	25283
76	weights	23946
77	provided	23704
78	gptq	23673
79	bias	23479
80	hardware	23088
81	downstream	22466
82	new	22437
83	citation	22272
84	run	22096
85	size	21942
86	generated	21868
87	started	21840
88	compute	21814
89	first	21671
90	architecture	21661
91	install	21629
92	note	21613
93	testing	21398
94	risks	21295
95	llama	20854
96	gguf	20693
97	click	20490
98	nan	20408
99	thanks	20240
100	unknown	20194

Appendix 3a: Code for data collection

```python
import pandas as pd
import os
import concurrent.futures
import pyarrow as pa
import pyarrow.parquet as pq

# Import HF API & login using token
import huggingface_hub
from huggingface_hub import HfApi, ModelCard, list_models

hf_api = HfApi(
    endpoint="https://huggingface.co",
    token="hf_xxx",
)

# Retrieve list of models
modelsD = hf_api.list_models(full=True, cardData=True, sort='downloads', direction=-1)
modellistD = list(modelsD)
print(len(modellistD))

# Set directories
model_card_directory = ""
parquet_directory = ""

# function to download model cards & create empty ones where there are none
def process_model(model_id, model_card_directory):
    filename = model_id.replace('/', "'")
    fullpath = os.path.join(model_card_directory, filename)

    try:
        modelcard = ModelCard.load(model_id, ignore_metadata_errors=True)
        with open(fullpath, 'w', encoding='utf-8') as file:
            file.write(modelcard.content)
    except Exception as e:
        if "404" in str(e):
            with open(fullpath, 'w', encoding='utf-8') as file:
                file.write("")
        else:
            print(f"Error processing model ID {model_id}: {str(e)}")

# function to process batches in parallel for efficiency
def process_batch(batch, directory):
    for model in batch:
        process_model(model.id, directory)

batch_size = 1000
batches = [modellistD[i:i+batch_size] for i in range(0, len(modellistD), batch_size)]

with concurrent.futures.ThreadPoolExecutor() as executor:
    futures = [executor.submit(process_batch, batch, model_card_directory) for batch in batches]
    concurrent.futures.wait(futures)

# create the master pandas df and store metadata
data = []
for model in modellistD:
    # Extract dataset information from tags
    datasets = [tag.split(':')[-1] for tag in model.tags if tag.startswith('dataset:')]
    if not datasets:
        datasets = ['']

    data.append([
        model.id,
        model.author,
        model.created_at,
        model.downloads,
        model.likes,
        model.pipeline_tag,
        datasets
    ])

headers = ['id', 'author', 'date published', 'downloads', 'likes', 'tags', 'datasets']
df = pd.DataFrame(data, columns=headers)

# convert the df to and Arrow table & store to the master parquet file
table = pa.Table.from_pandas(df)
parquet_file = os.path.join(parquet_directory, '240327_all.parquet')
pq.write_table(table, parquet_file)

print("done")
```

Appendix 3b: Code for cleaning and parsing model cards

```
import mistune
import os
import re
from bs4 import BeautifulSoup

# create custom mistune renderer used to remove tables, code, links, and images & save as a .txt file
# used HTMLRenderer because mistune.renderers.markdown.MarkdownRenderer did not exist

class CustomRenderer(mistune.HTMLRenderer):
    def block_code(self, code, info=None): # delete code blocks
        return ''

    def block_html(self, text): # delete HTML tables or images
        if text.startswith('<table') or text.startswith('<img') or text.startswith('<a href='):
            return ''
        else:
            return super().block_html(text)

    def link(self, text, url, title=None): # delete links
        return text

    def image(self, src, url, alt="", title=None): # delete images
        return ''

    def table(self, header, body): # delete tables
        return ''

    def render(self, text):
        output = super().render(text)
        output = output.replace('&quot;', '"') # decode HTML entities for double quotes
        return output

# function for parsing
def parse2txt(markdown_file, output_folder):
    try:
        with open(markdown_file, 'r', encoding='utf-8') as file:
            markdown_content = file.read()
    except UnicodeDecodeError as e:
        print(f"Error decoding file {markdown_file}: {e}. Skipping.")
        return

    # split content and only keep the body of the text (remove metadata etc)
    parts = markdown_content.split('---', 2)
    if len(parts) == 3:
        markdown_content = parts[2]

    # remove other formatted markdown tables from markdown_content
    markdown_content = re.sub(r'^\|.*\|$\n(?:\|:-?.*\|.*\n)*\n?', '', markdown_content, flags=re.MULTILINE)

    #apply the renderer to the content
    parsedMC = mistune.create_markdown(renderer=CustomRenderer())(markdown_content)

    # remove html tags
    soup = BeautifulSoup(parsedMC, 'html.parser')
    pure_txt = soup.get_text()
    pure_txt = pure_txt.replace('<p>', '\n\n').replace('</p>', '') # HTML paragraphs -> new-line
    for tag in soup.find_all(['h1', 'h2', 'h3', 'ul', 'li', 'code']):
        tag.replace_with(' ')
    pure_txt = BeautifulSoup(pure_txt, 'html.parser').get_text()

    # create path for outputted .txt files
    parsed_txt_file = os.path.join(output_folder, f"{os.path.basename(markdown_file)}.txt")

    # write the cleaned text to the .txt file
    with open(parsed_txt_file, 'w') as output:
        output.write(pure_txt)

    return pure_txt

input_folder = ""
output_folder = ""

# loop through all the files in the input folder, applying the parse2txt function to each of them
for filename in os.listdir(input_folder):
    try:
        markdown_file = os.path.join(input_folder, filename)
        if os.path.isfile(markdown_file):
            parse2txt(markdown_file, output_folder)
        else:
            print(f"Skipping: {markdown_file}")
    except MemoryError as e:
        print(f"MemoryError occurred while processing file: {filename}. Skipping.")
        continue
    except Exception as e:
        print(f"An error occurred while processing file: {filename}. Skipping.")
        continue

print("done")
```

Appendix 3c: Code for counting most common words using nltk

```
import os
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import Counter
import pandas as pd

directory = ''

# initialize counter to store values
word_counter = Counter()

# iterate through each .txt model card in the directory
for filename in os.listdir(directory):
    if filename.endswith('.txt'):
        file_path = os.path.join(directory, filename)
        with open(file_path, 'r', encoding='utf-8') as file:
            # read files & tokenize text
            text = file.read()
            words = word_tokenize(text)
            # filter out stopwords
            stop_words = set(stopwords.words('english'))
            words = [word.lower() for word in words if word.isalpha() and word.lower() not in stop_words]
            # update word counter
            word_counter.update(words)

# select top 100 most common words
top_1000_words = word_counter.most_common(1000)

# save to a df and then a csv
df = pd.DataFrame(top_1000_words, columns=['Word', 'Frequency'])
df.to_csv('')
```

Appendix 3d: Code for keyword, heading, and word count

```
import pandas as pd
import os

# Read the pq file into a df
df = pd.read_parquet('')

# parsed model card .txt file directory
modelcard_dir = ''

# keywords & headings
keywords = ['bias', 'risks', 'limitations', 'biases', 'biased', 'risk', 'limitation', 'fairness', 'caveat',
'caveats', 'problematic', 'harmful', \
            'discrimination', 'discriminatory', 'stereotype', 'stereotypes', 'stereotypical', 'race', 'racist',
'intersectional', 'prejudice', 'prejudices', \
            'inequality', 'inequalities','out of scope use', 'out-of-scope use', 'out of scope uses', 'out-of-scope
uses', 'misuse', 'misused',\
            'environmental impact', 'environmental impacts', 'carbon emissions', 'water', 'danger', 'dangerous',
'disinformation', 'misinformation', 'cybersecurity', \
            'abuse', 'sexual abuse', 'pornography', 'pornographic', 'scam', 'scams', 'mislead', 'misleading',
'untrue', 'Bias, Risks, and Limitations', ]

headings = ['Bias, Risks, and Limitations', "Bias, Risks, and Limitations\n\n[More Information Needed]", \
            'Out-of-Scope Use', "Out-of-Scope Use\n\n[More Information Needed]", "This model card has been
automatically generated."]

# error log
error_log_path = ''

# iterate through each row in the df
for index, row in df.iterrows():
    txt_file_name = row['id'].replace('/', "'") + '.txt'  # generate the file name from id replacing / with '
    txt_file_path = os.path.join(modelcard_dir, txt_file_name)  # generate file path

    try:
        with open(txt_file_path, 'r', encoding='utf-8') as file:

            content = file.read()

            #check if file is blank
            if len(content) == 0:
                df.at[index, 'modelcard'] = False
            else:
                df.at[index, 'modelcard'] = True

            # count words in each file
            word_count = len(content.split())
            df.at[index, 'wordcount'] = word_count

            # set keyword count for each file at 0
            totalkw_count = 0

            # count occurrences of each heading in the content (case-sensitive)
            for heading in headings:
                found = heading in content
                df.at[index, f'{heading}_val'] = found

            # convert all content to lowercase & count occurrences of each keyword
            contentL = content.lower()
            for keyword in keywords:
                count = contentL.count(keyword.lower())
                df.at[index, f'{keyword}_count'] = count
                totalkw_count += count

            # store total keyword count for each file in a new column 'total_kw'
            df.at[index, 'totalkw'] = totalkw_count

    except FileNotFoundError:
        with open(error_log_path, 'a') as error_log:
            error_log.write(f"File not found: {row['id']}\n")
            is_blank = True  # assume file is blank if not found
        continue

# write the modified df back to a new pq file
df.to_parquet('', index=False)

print('done')
```

Appendix 3e: Code for data analysis – general trends and word count

Load master parquet file as a pandas df and create other dfs ranked by different metrics + equivalents excluding all models without model cards

```
data_df = pd.read_parquet('')

downloads_df = data_df.sort_values(by='downloads', ascending=False) #sorted by downloads
dates_df = data_df.sort_values(by='date published', ascending=False) #sorted by date published (most recent)
likes_df = data_df.sort_values(by='likes', ascending=False)  #sorted by likes
has_words_df = data_df[data_df['wordcount'] != 0]  #eliminating empty cards

has_words_downloads_df = downloads_df[downloads_df['wordcount'] != 0]  #eliminating empty cards
has_words_date_df = dates_df[dates_df['wordcount'] != 0]
has_words_likes_df = likes_df[likes_df['wordcount'] != 0]
```

Figure 1a

```
# convert 'date published' datetime format
data_df['date published'] = pd.to_datetime(data_df['date published'])

# group data by date published and count number of cards published per day for each category
empty_df = data_df[data_df['wordcount'] == 0]
non_empty_df = data_df[data_df['wordcount'] > 0]

cards_per_day_empty = empty_df.groupby(empty_df['date published'].dt.date).size()
cards_per_day_non_empty = non_empty_df.groupby(non_empty_df['date published'].dt.date).size()

# calculate total number of models published per day
total_models_per_day = data_df.groupby(data_df['date published'].dt.date).size()

# cumulative sum for each category
cumulative_empty = cards_per_day_empty.cumsum()
cumulative_non_empty = cards_per_day_non_empty.cumsum()
cumulative_total = total_models_per_day.cumsum()

plt.figure(figsize=(12, 8))

cumulative_empty.plot(marker='', linestyle='-', label='Models with No/Empty Model Cards', color='deepskyblue')
cumulative_non_empty.plot(marker='', linestyle='-', label='Models with Model Cards', color='salmon')
cumulative_total.plot(marker='', linestyle='-', label='Total Models', color='teal')

plt.title('Cumulative Number of Models Published on Hugging Face Over Time (2022-2024)')
plt.xlabel('Date')
plt.ylabel('Cumulative Number of Models Published')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 1b

```
# convert 'date published' datetime format
data_df['date published'] = pd.to_datetime(data_df['date published'])

# group data by date published and count number of cards published per day for each category
empty_df = data_df[data_df['wordcount'] == 0]  # Eliminating all except empty cards
non_empty_df = data_df[data_df['wordcount'] > 0]  # Eliminating empty cards
cards_per_day_empty = empty_df.groupby(empty_df['date published'].dt.date).size()
cards_per_day_non_empty = non_empty_df.groupby(non_empty_df['date published'].dt.date).size()

# calculate total models published per day
total_cards_per_day = cards_per_day_empty.add(cards_per_day_non_empty, fill_value=0)

# rolling mean to smooth
rolling_mean_empty = cards_per_day_empty.rolling(window=7).mean()  # Adjust the window size as needed
rolling_mean_non_empty = cards_per_day_non_empty.rolling(window=7).mean()  # Adjust the window size as needed
rolling_mean_total = total_cards_per_day.rolling(window=7).mean()  # Adjust the window size as needed

plt.figure(figsize=(12, 8))

rolling_mean_empty.plot(marker='', linestyle='-', label='Models with Empty Model Cards', color= 'deepskyblue')
rolling_mean_non_empty.plot(marker='', linestyle='-', label='Models with Non-empty Model Cards', color='salmon')
rolling_mean_total.plot(marker='', linestyle='-', label='Total Models', color='teal')

plt.title('Number of Models Published per Day (Rolling 7-Day Mean), 2022-2024')
plt.xlabel('Date')
plt.ylabel('Number of Cards Published')
plt.legend()
plt.ylim(0,2000)
plt.grid(True)
plt.show()
```

Figure 2a.

```
plt.figure(figsize=(20, 6))
sns.displot(has_words_df['wordcount'], bins=50000, color='orchid', height=6, aspect=2)
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Distribution of Non-Zero Word Count Values')
plt.xlim(0, 1000)  # Set the x-axis limit
plt.show()
```

Figure 2b.

```
#IQR + outlier bounds
q25, q75 = has_words_df['wordcount'].quantile([0.25, 0.75])
iqr = q75 - q25

lower_bound = q25 - 1.5 * iqr
upper_bound = q75 + 1.5 * iqr

# Filter DataFrame to remove outliers
wc_outliers_removed_df = has_words_df[(has_words_df['wordcount'] >= lower_bound) & (has_words_df['wordcount'] <=
upper_bound)]

# distribution of non-outlier values
plt.figure(figsize=(20, 6))
plt.hist(wc_outliers_removed_df['wordcount'], bins=29, color='orchid')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Distribution of Non-Zero and Non-Outlier Values')
plt.show()
```

Figure 3a.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# generate chunks
chunk_size = 10000
chunks = [downloads_df[i:i+chunk_size] for i in range(0, len(downloads_df), chunk_size)]

# calculate mean word count & total downloads for each chunk
mean_wordcounts = [chunk['wordcount'].mean() for chunk in chunks]
total_downloads = [chunk['downloads'].sum() for chunk in chunks]

fig, ax1 = plt.subplots(figsize=(18, 6))

# Plot the bar chart for mean word counts on the left y-axis
ax1.bar(range(len(mean_wordcounts)), mean_wordcounts, color='skyblue', label='Mean word count', alpha=0.5)
ax1.set_xlabel('Chunk (10,000 models)')
ax1.set_ylabel('Mean word count')
ax1.set_title('Mean word count and total download count, by chunk of 10,000 grouped by downloads (descending)')
ax1.set_xticks(range(len(mean_wordcounts)))
ax1.set_xticklabels([f'{i+1}' for i in range(len(mean_wordcounts))])
ax1.set_ylim(0, 250)

# fit polynomial regression model to mean word count + plot
X = np.arange(len(mean_wordcounts)).reshape(-1, 1)
y = np.array(mean_wordcounts).reshape(-1, 1)
degree = 2
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
model_poly = LinearRegression()
model_poly.fit(X_poly, y)
trend_line_poly = model_poly.predict(X_poly)
r_squared_poly = r2_score(y, trend_line_poly)

ax1.plot(trend_line_poly, color='salmon', linestyle='--', label=f'Polynomial regression of mean word count
(R^2={r_squared_poly:.4f})')

# second y-axis on the right for total download count
ax2 = ax1.twinx()
ax2.bar(range(len(total_downloads)), total_downloads, color='orange', label='Total download count', alpha=0.4)
ax2.set_ylabel('Total download count (log)')
ax2.set_yscale('log')

# create single legend
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='upper right')
plt.show()
```

Figure 3b

```
chunk_size = 1000
sorted_downloads_10k_df = downloads_df.head(10000)
chunks = [sorted_downloads_10k_df[i:i+chunk_size] for i in range(0, len(sorted_downloads_10k_df), chunk_size)]

# calculate mean word count for each chunk
mean_word_counts = [chunk['wordcount'].mean() for chunk in chunks]

plt.figure(figsize=(11, 8))
plt.bar(range(len(mean_word_counts)), mean_word_counts, color='skyblue')
plt.xlabel('Chunk (1000 models)')
plt.ylabel('Mean word count')
plt.title('Mean word count of top 10,000 models, by chunk of 1,000 most downloaded')
plt.xticks(range(len(mean_word_counts)), [f'{i+1}' for i in range(len(mean_word_counts))])
plt.show()
```

Figure 3c

```
# sort by 10,000 chunks but only taking MCs with content

downloadsmc_df = has_words_df.sort_values(by='downloads', ascending=False)

chunk_size = 10000
chunks = [downloadsmc_df[i:i+chunk_size] for i in range(0, len(downloadsmc_df), chunk_size)]

# calculate mean word count for each chunk
mean_wordcounts = [chunk['wordcount'].mean() for chunk in chunks]

plt.figure(figsize=(8, 8))
plt.bar(range(len(mean_wordcounts)), mean_wordcounts, color='skyblue')
plt.xlabel('Chunk (10,000)')
plt.ylabel('Mean word count')
plt.title('Mean word count, by chunk of 10,000 most downloaded (models with model cards only)')
plt.xticks(range(len(mean_wordcounts)), [f'{i+1}' for i in range(len(mean_wordcounts))])
plt.show()
```

Figure 4a

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming your DataFrame is named data_df
sorted_date_df = data_df.sort_values(by='date published', ascending=True)

# Set the 'date published' column as the index if it's not already
sorted_date_df.set_index('date published', inplace=True)

# Resample the DataFrame to monthly frequency and calculate mean word count
monthly_mean_word_counts = sorted_date_df.resample('ME')['wordcount'].mean()

# Plotting
plt.figure(figsize=(15, 6))
monthly_mean_word_counts.plot(kind='bar', color='salmon', width=0.9)
plt.xlabel('Date')
plt.ylabel('Mean word count')
plt.title('Monthly mean word count (all models)')



# Extracting tick positions and labels
tick_positions = []
tick_labels = []
prev_year = None
for i, date in enumerate(monthly_mean_word_counts.index):
    current_year = date.year
    current_month = date.strftime('%b')
    if current_year != prev_year:
        tick_positions.append(i)
        tick_labels.append(f"{current_month} {current_year}")
        prev_year = current_year
    else:
        tick_positions.append(i)
        tick_labels.append(current_month)

# Setting ticks and labels
plt.xticks(tick_positions, tick_labels, rotation=45, ha='right')
plt.tight_layout()  # Adjust layout to prevent clipping of tick labels
plt.show()
```

Figure 4b

```
# Resample DataFrame to monthly frequency and calculate mean word counts

sorted_date_df = has_words_df.sort_values(by='date published', ascending=True)
sorted_date_df.set_index('date published', inplace=True)
monthly_mean_word_counts = sorted_date_df.resample('ME')['wordcount'].mean()

# Do the same for 99th percentile
has_words99_df = has_words_df[has_words_df['wordcount'] <= pc99]
sorted_date_df = has_words99_df.sort_values(by='date published', ascending=True)
sorted_date_df.set_index('date published', inplace=True)
monthly_mean_word_counts99 = sorted_date_df.resample('ME')['wordcount'].mean()

# Filter the DataFrame to keep values below the 99th percentile
has_words975_df = has_words_df[has_words_df['wordcount'] <= pc975]
sorted_date_df = has_words975_df.sort_values(by='date published', ascending=True)
sorted_date_df.set_index('date published', inplace=True)
monthly_mean_word_counts975 = sorted_date_df.resample('ME')['wordcount'].mean()

# again for IQR
sorted_date_df = wc_outliers_removed_df.sort_values(by='date published', ascending=True)
sorted_date_df.set_index('date published', inplace=True)
monthly_mean_word_countsIQR = sorted_date_df.resample('ME')['wordcount'].mean()

# Convert dates to strings representing the month and year
monthly_mean_word_counts.index = monthly_mean_word_counts.index.strftime('%b %Y')
monthly_mean_word_counts99.index = monthly_mean_word_counts99.index.strftime('%b %Y')
monthly_mean_word_counts975.index = monthly_mean_word_counts975.index.strftime('%b %Y')
monthly_mean_word_countsIQR.index = monthly_mean_word_countsIQR.index.strftime('%b %Y')

# Plotting
plt.figure(figsize=(15, 6))

# Plot the bar charts for mean word counts with transparency
bar_width = 0.9  # Width of each bar
plt.bar(monthly_mean_word_counts.index, monthly_mean_word_counts, color='salmon', width=bar_width, alpha=1,
label='No outliers removed')
plt.bar(monthly_mean_word_counts99.index, monthly_mean_word_counts99, color='black', width=bar_width, alpha=0.2,
label='top 1% removed')
plt.bar(monthly_mean_word_counts975.index, monthly_mean_word_counts975, color='firebrick', width=bar_width,
alpha=0.5, label='top 2.5% removed')
plt.bar(monthly_mean_word_countsIQR.index, monthly_mean_word_countsIQR, color='skyblue', width=bar_width,
alpha=0.5, label='IQR method outliers removed')

plt.xlabel('Date')
plt.ylabel('Mean word count')
plt.title('Monthly mean word count (models with model cards)')
plt.legend()
plt.xticks(rotation=45, ha='right')
plt.tight_layout()  # Adjust layout to prevent clipping of tick labels
plt.show()
```

Appendix 3f: Code for data analysis – content

Figure 5a

```
import matplotlib.pyplot as plt

# First graph: percentage of models with non-zero wordcount
downloads_df = data_df.sort_values(by='downloads', ascending=False)  # sorted by downloads
chunk_size = 10000
chunks = [downloads_df[i:i + chunk_size] for i in range(0, len(downloads_df), chunk_size)]

percent_nonzero_wordcount = []
for chunk in chunks:
    nonzero_count = (chunk['wordcount'] != 0).sum()
    chunk_length = len(chunk)
    percentage = (nonzero_count / chunk_length) * 100
    percent_nonzero_wordcount.append(percentage)

# first graph (models with model card)
plt.figure(figsize=(18, 6))
plt.bar(range(len(percent_nonzero_wordcount)), percent_nonzero_wordcount, color='skyblue', alpha=1, label='% of
models with a model card')
plt.xlabel('Chunk (10,000)')
plt.ylabel('% of models')
plt.title('Percentage of models with a non-zero wordcount, by chunk of 10,000 most downloaded')
plt.xticks(range(len(percent_nonzero_wordcount)), [f'{i+1}' for i in range(len(percent_nonzero_wordcount))])
plt.ylim(0, 100)

# second graph: models with sociotechnical terms
percentages = []
for chunk in chunks:
    percentage = (chunk.merge(filtered_models, on='id', how='inner').shape[0] / len(chunk)) * 100
    percentages.append(percentage)

# plot second graph and overlay bars
plt.bar(range(len(percentages)), percentages, color='seagreen', alpha=1, label='% of models including any
sociotechnical keyword')

plt.legend()
plt.show()
```

Figure 5b

```
import matplotlib.pyplot as plt

chunk_size = 100
sorted_downloads_10k_df = downloads_df.head(1000)
chunks = [sorted_downloads_10k_df[i:i+chunk_size] for i in range(0, len(sorted_downloads_10k_df), chunk_size)]


percentages = []

# calculate percentage of rows found in sociotechnical df for each chunk
for chunk in chunks:
    percentage = (chunk.merge(filtered_models, on='id', how='inner').shape[0] / len(chunk)) * 100
    print(percentage)
    percentages.append(percentage)

plt.figure(figsize=(12, 6))
plt.bar(range(1, len(percentages) + 1), percentages, color='seagreen')
plt.xlabel('Chunk (1,000)')
plt.ylabel('% of models including any sociotechnical keyword')
plt.title('Percentage of top 20,000 models including any sociotechnical keyword, by chunk of 1,000 most
downloaded')
plt.xticks(range(1, len(percentages) + 1), [f'{i}' for i in range(1, len(percentages) + 1)])
plt.show()
```

Figure 5c

```
# Group by month and count the number of models published per month
models_per_month = has_words_date_df.groupby(has_words_date_df['date published'].dt.to_period('M')).size()

percentages = []

# calculate sociotechnical presence percentage per month
for month, group in has_words_date_df.groupby(has_words_date_df['date published'].dt.to_period('M')):
    common_models_count = len(group.merge(filtered_models, on='id', how='inner'))

    # Calculate the percentage for the current month
    percentage_in_filtered_models = (common_models_count / len(group)) * 100
    percentages.append(percentage_in_filtered_models)

# format x-axis labels
def format_month_labels(x):
    month = x.strftime('%b %Y')
    return month

format_month_labels.last_year = None

# Plotting
plt.figure(figsize=(15, 6))
plt.bar(models_per_month.index.to_timestamp(), percentages, color='salmon', width=25, align='center')
plt.xlabel('Month')
plt.ylabel('% of models including any sociotechnical keyword')
plt.title('Percentage of models with a model card including any sociotechnical keyword, by month')

# Customize x-axis labels using the format_month_labels function
plt.xticks(models_per_month.index.to_timestamp(), [format_month_labels(x) for x in models_per_month.index],
rotation=45)

plt.tight_layout()
plt.show()
```