Write Code that People Want to Use

Anonymous Authors[1]

Abstract

"Research code" is a common, often self-effacing, term used to refer to the type of code that is commonly released alongside research papers. Research code is notorious for being fragile, poorly documented, and difficult for others to run or extend. In this position paper, we argue that, while research code seems to meet the short-term needs of research projects, in fact the practice hurts researchers by limiting the impact of their work and causing fewer people to build on their research. We explore the structural incentives and dynamics of the field that drive these behaviors. We argue that extensibility matters far more than strict reproducibility for research impact, and propose both pragmatic approaches for individual researchers and institutional reforms to encourage the development of more usable and maintainable research software.

1. Introduction

Modern ML research is highly oriented towards developing on software artifacts: new algorithms, models, benchmarks, tools, etc. However, the norms and practices surrounding software development in research contexts often prioritize expedience over enduring functionality. The result is a proliferation of what is colloquially referred to as "research code": scripts and codebases written for the purpose of generating publishable results, but seldom designed with reuse or extensibility in mind. Such code is frequently undocumented, brittle, and narrowly tailored to a specific experimental configuration.

If a core output of machine learning research is writing code, we should write it with others in mind. While reproducibility is often cited as the reason to release code, most researchers are not interested in re-running someone else's experiment—they want to build on it. For this purpose, extensibility matters far more than strict reproducibility. Code that can be adapted, modified, and extended enables broader adoption and amplifies research impact.

This paper advocates for a cultural shift in ML research to emphasize usable and extensible software development. We argue that this shift is essential for improving the efficiency of the field, increasing the propagation of ideas and adoption of new methods. We explore the dynamics of the field that drive these behaviors and distill learnings from over 150 papers, building on seven major research papers to identify best practices in research code that can encourage adoption and wider use.

2. What do we want to see in codebases?

2.1. Different Artifacts, Same Bottleneck

The intended impact of a work may differ depending on the type of work. For example, works introducing benchmarks or evaluations are considered highly impactful if they are frequently used in benchmarking new models or inspire the development of a new evaluation. Algorithms are considered highly impactful if future work utilizes these algorithms, develops extensions of them, or is otherwise inspired by the original method. Pre-trained models achieve impact when practitioners can drop them into existing systems without extensive engineering effort.

Despite these differences in how impact manifests, a single bottleneck dominates: *extensibility*. Easily configuring an evaluation to run on a new model, importing an algorithm from a well-documented repository, or loading a model through a familiar API all dramatically increase the likelihood that a contribution will be adopted and extended. The subsections below examine how this principle plays out for benchmarks and evaluations, algorithms, and model–inference pipelines in turn.

Benchmarks and Evaluations. The scientific value of a benchmark is not measured by its citation count alone, but by how often it is used. Making a benchmark easy for people to run on their models, therefore, is essential. Classic examples include GLUE in NLP (Wang et al., 2019) and Atari (Bellemare et al., 2013) in RL, both of which owe their ubiquity to straightforward, well-supported reference implementations.

BIG-Bench (Srivastava et al., 2023) and BIG-Bench Hard (Suzgun et al., 2022) illustrates how implementation choices affect practical adoption. Despite BIG-Bench representing an impressive collaborative effort by over 400 researchers,

its coupling to Google's internal infrastructure made it difficult for most researchers to use. BIG-Bench Hard, a curated subset of 23 challenging tasks, has seen broader adoption, but tellingly, researchers consistently bypass the official codebase entirely[1]. Instead, they either re-implement the evaluation logic from scratch or integrate the tasks into existing frameworks like the Eval Harness (Gao et al., 2024). When code isn't designed for extensibility, even valuable contributions require duplicated engineering effort across the field.

Algorithms. Algorithmic papers achieve long-term impact when later work reimplements, extends, or mathematically analyses the method. In practice, adoption is gated by the *first* public implementation that researchers can easily import or hack rather than re-code.

Models and Inference Pipelines. When a new pretrained model comes out, people often want to be able to experiment with it right away. When there is a unified set of expectations for the model API and inference frameworks are flexible enough, new models can be drop-in replacements for old models. However, when inference codebases are overfit to specific model details or the field lacks a set of expectations for model behavior, there is a substantial pain period while people work to figure out how to get a new model to run in existing pipelines.

At it's core, the most important question for adoption is: "how many engineering hours stand between a curious reader and a working extension?" The more we can smooth the user experience and create systems that empower people to do research, the more impactful our research will be.

2.2. Research Infrastructure

Research infrastructure can alleviate burdens associated with utilizing research code and facilitate the usage of research artifacts. For cases such as benchmarks, evaluations, and certain algorithms, reimplementing even the most well-documented code can require many engineering hours, potentially hindering future usage. As the friction to extend an existing work increases, the likelihood the work achieves the researchers' desired impact decreases.

Many types of work face similar issues to adoption. Configuring evaluations to run on a variety of models without proper tooling creates barriers, as does repeatedly duplicating an algorithm's implementation to be utilized. Shared infrastructure can help alleviate these challenges. For example, the transformers library from Hugging Face (Wolf et al., 2020) facilitates easy access to datasets and models with minimal reconfiguration for environmental differences, EleutherAI's Language Model Eval Harness (Gao et al., 2024) supports running evaluations on numerous open-source and proprietary models through a unified API, and PyTorch (Paszke et al., 2019) facilitates development of architectures, algorithms, and models based on tensors.

Shared infrastructure not only eases adoption but improves the quality and comparability of research outputs across the field. Having a shared implementation utilized across various works not only reduces research time, as common baselines do not need to be reimplemented for each project, but reduces the introduction of unnecessary variables and variance between implementations of previous work that make scientific comparison challenging and can undermine the reliability of scientific findings. Integrating research artifacts into established research infrastructure—for example, incorporating a newly developed evaluation into the Eval Harness—can facilitate broader adoption and usage of the evaluation or other artifacts added to existing libraries (Biderman et al., 2024).

2.3. Between Research Code and Research Infrastructure

Although research infrastructure greatly benefits researchers who contribute to it and the research community, not all work can or should strive to produce new research infrastructure. Well-documented, well-designed, stand-alone code bases are more than sufficient to meet the needs we articulate above for many researchers. In such cases, we find that *hackability* and *flexibility* are often key to enabling future researchers to build on previous work.

Another option for bridging the gap between ad-hoc research code and full-fledged infrastructure is to contribute new functionality back to *existing* ecosystems rather than publishing an isolated repository. Adding a task to the LM Evaluation Harness (Gao et al., 2024), registering a new optimizer in Stable-Baselines3 (Raffin et al., 2021), or upstreaming a data-loader to torchvision (Paszke et al., 2019) immediately places the contribution behind a stable API, inherits continuous integration and community maintenance, and eliminates redundant boilerplate. In practical terms, this means a graduate student looking to replicate or extend the work can do so with a single import statement instead of cloning yet another bespoke repo. Such in-framework extensions retain the hackability of stand-alone code—one can still experiment locally—while gaining the discoverability,

[1]To reach this conclusion, we surveyed 50 papers that cite BBH chosen uniformly at random. Twenty-five of those papers evaluate on BBH, and we could not confirm any examples that use the official codebase. Of those 25 papers, 4 use lm-eval, 8 explicitly use a custom implementation, and 13 do not specify. In every case, the papers that do not specify use an inference methodology or model architecture that is incompatible with the official codebase, though we cannot rule out that they modified the official code to accomplish this. We tried to survey BigBench, but zero out of the first twenty papers we looked at evaluated on BigBench.

versioning discipline, and longevity that characterize mature research infrastructure.

3. Why are we struggling to do this

Most research code falls very far short of the goals outlined in the previous section. Researchers routinely reimplement algorithms, models, and methodologies from scratch rather than building on existing code. This section examines the structural barriers that perpetuate these practices.

3.1. Fundamental Barriers to Code Quality

Incentives and the publication race The primary currency of academics is publication in peer-reviewed conferences and journals, and the citations they accumulate. While the most successful and widely adopted infrastructure can eventually garner significant citations (the Eval Harness has over 1,000 and Stable-Baselines3 has over 3,000), this represents a substantial, multi-year effort. Research code is often described as meeting the bare minimum for viability, a reflection on the fact that released code often is held to the standard of not completely embarrassing the authors and little more. Consequently, the system generally prioritizes research outputs that can demonstrate results quickly. This typically means code optimized for reproducing the specific findings of a paper rather than being engineered for broader extensibility and long-term use by others.

Code as means, not output Despite some positive movements towards valuing code as a meaningful research contribution, including this amazing workshop, the quality of research code is rarely considered in high-impact, peer-reviewed venues. While many venues encourage reviewers to look at released code, we are unaware of any major machine learning venue that expects reviewers to judge research in part on the quality of the released code. This disconnect manifests in review processes that prioritize algorithmic novelty and empirical results while systematically ignoring code quality, usability, and potential for community impact. When a paper's impact is measured exclusively in citations rather than GitHub stars or downstream usage, investing in code quality appears irrational.

3.2. Academic Challenges

Some challenges are unique to or more pronounced in academic research contexts.

Lack of training Academic training compounds this problem by treating programming as a research tool rather than a software discipline. With curricula rarely integrating software engineering practices and advisors seldom enforcing code standards, researchers lack the foundational skills needed for writing maintainable, extensible code.

Turnover The high turnover rate in academic settings (e.g., students graduating, postdocs moving on) makes maintaining research infrastructure particularly challenging. Maintaining extensible tools and updating them in response to changes in the field is a long-term commitment which is difficult to sustain when key personnel and their specific knowledge depart. The deep, often tacit, knowledge associated with a codebase can be lost when key developers depart, leaving projects vulnerable to stagnation or abandonment, regardless of their initial quality or potential utility as extensible infrastructure.

3.3. Industry Challenges

Industry faces different but equally significant barriers.

Internal code bases Code developed within companies is often deeply coupled to proprietary infrastructure, making extraction and generalization prohibitively expensive. Even well-intentioned releases can fail when internal assumptions pervade the codebase. BigBench (Srivastava et al., 2023), a massive collaborative benchmark created by over 400 researchers, illustrates this perfectly: despite being designed for the entire community, its implementation was so tied to Google's internal infrastructure that it could not run most models from HuggingFace (the de facto standard) without substantial community effort to rebuild it.[2] Consequently, the published paper does not evaluate BigBench on any publicly accessible language model.

Moreover, internal incentive misalignment creates friction. Research teams are evaluated on product impact and novel capabilities, not on enabling external extensibility that could benefit competitors. The engineering effort required to decouple code from internal systems rarely aligns with quarterly objectives or promotion criteria.

4. How we can do better: Insights and Recommendations

Addressing challenges with developing extensible research code and research infrastructure requires both individual researcher action and systematic change. In this section, we outline pragmatic approaches researchers can adopt to develop research infrastructure and code that prioritizes usability in Section 4.1, as well as institutional changes the ML community should adopt to encourage and prioritize the development of research infrastructure over research code in Section 4.2.

[2] See GitHub issue #551 and pull request #830

4.1. Pragmatic Approaches for Researchers

While writing this paper, we surveyed at least twenty papers building on or using each of the projects we have discussed so far: PyTorch, lm-eval, transformers, BigBench, GLUE, Stable-Baselines3, and torchvision. In this section, we distill learnings about the commonalities that drive the adoption of widely successful research infrastructure.

Modularization Our studies demonstrate that research impact correlates strongly with code extensibility. Researchers should consider potential adaptations during initial development and prioritize modular design at obvious points of divergence. This approach yields immediate returns: modular, well-structured code accelerates your own experimentation, particularly when addressing reviewer feedback or pursuing follow-up work.

Consider Integration Paths While standalone implementations serve an important exploratory function, particularly for theoretical contributions, negative results, or highly specialized analyses, researchers should evaluate whether their contributions align with existing community infrastructure. Integration decisions can be guided by assessing whether established frameworks address the majority of required functionality, thereby reducing implementation overhead while increasing adoption potential. For instance, evaluation benchmarks integrated into frameworks like the Eval Harness demonstrate significantly higher utilization rates than standalone implementations (Biderman et al., 2024).

Documentation Effective documentation should prioritize design rationale and extensibility interfaces over procedural installation instructions. This approach reduces barriers to community adoption while decreasing maintenance burden and improved code comprehension during future development. Stable-Baselines3 (Raffin et al., 2021) exemplifies this approach: by documenting architectural decisions and modification protocols, the authors enabled both community contributions and their own continued development efficiency.

4.2. Institutional Reforms

Recognize Code Contributions in Evaluation Metrics Current assessment focuses on publications and citations while ignoring code impact. Institutions should value well-documented, extensible code releases alongside traditional outputs. When a clean PyTorch implementation enables dozens of follow-up papers, that contribution merits recognition. Venues should assess code quality as part of the review process, not merely require code availability.

Sustainable Maintenance Models Creating widely used infrastructure can become a burden when maintenance falls entirely on creators. We need funding models that acknowledge this reality: dedicated infrastructure postdocs, community maintenance grants, or transition pathways where creators can hand off maintenance while retaining credit. Programs such as NSF's POSE[3] and the Linux Foundation's stewardship model[4] provide a template, but coverage remains sparse relative to need.

5. Conclusion

Extensible, well-documented software is now a prerequisite for durable impact in machine-learning research. Our survey of more than 150 downstream papers demonstrates a consistent pattern: contributions purposely engineered for hackability are reused, extended, and cited far more often than equally novel ideas released as self-contained scripts. This finding underscores that code quality is not ancillary to scientific merit; it is part of the result itself.

We therefore advocate three concrete changes. First, authors should treat extensibility as a design requirement, providing modular APIs and narrative documentation that explain why architectural decisions were made. Second, reviewers and program committees should incorporate explicit software-quality criteria into their rubrics, recognizing that reproducibility is a baseline but that true impact depends on ease of adaptation. Third, research organizations must invest in the long-term maintenance of community infrastructure because unmaintained code rapidly accrues technical debt. By embedding these norms, the field can ensure that new ideas travel farther and accelerate collective progress, turning individual repositories into shared platforms for discovery.

References

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Biderman, S., Schoelkopf, H., Sutawika, L., Gao, L., Tow, J., Abbasi, B., Aji, A. F., Ammanamanchi, P. S., Black, S., Clive, J., et al. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika,

[3]https://www.nsf.gov/funding/opportunities/pose-pathways-enable-open-source-ecosystems

[4]https://www.linuxfoundation.org/projects/hosting

L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research*, 22(268):1–8, 2021.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., Kluska, A., Lewkowycz, A., Agarwal, A., Power, A., Ray, A., Warstadt, A., Kocurek, A. W., Safaya, A., Tazarv, A., Xiang, A., Parrish, A., Nie, A., Hussain, A., Askell, A., Dsouza, A., Slone, A., Rahane, A., Iyer, A. S., Andreassen, A., Madotto, A., Santilli, A., Stuhlmüller, A., Dai, A., La, A., Lampinen, A., Zou, A., Jiang, A., Chen, A., Vuong, A., Gupta, A., Gottardi, A., Norelli, A., Venkatesh, A., Gholamidavoodi, A., Tabassum, A., Menezes, A., Kirubarajan, A., Mullokandov, A., Sabharwal, A., Herrick, A., Efrat, A., Erdem, A., Karakaş, A., Roberts, B. R., Loe, B. S., Zoph, B., Bojanowski, B., Özyurt, B., Hedayatnia, B., Neyshabur, B., Inden, B., Stein, B., Ekmekci, B., Lin, B. Y., Howald, B., Orinion, B., Diao, C., Dour, C., Stinson, C., Argueta, C., Ramírez, C. F., Singh, C., Rathkopf, C., Meng, C., Baral, C., Wu, C., Callison-Burch, C., Waites, C., Voigt, C., Manning, C. D., Potts, C., Ramirez, C., Rivera, C. E., Siro, C., Raffel, C., Ashcraft, C., Garbacea, C., Sileo, D., Garrette, D., Hendrycks, D., Kilman, D., Roth, D., Freeman, D., Khashabi, D., Levy, D., González, D. M., Perszyk, D., Hernandez, D., Chen, D., Ippolito, D., Gilboa, D., Dohan, D., Drakard, D., Jurgens, D., Datta, D., Ganguli, D., Emelin, D., Kleyko, D., Yuret, D., Chen, D., Tam, D., Hupkes, D., Misra, D., Buzan, D., Mollo, D. C., Yang, D., Lee, D.-H., Schrader, D., Shutova, E., Cubuk, E. D., Segal, E., Hagerman, E., Barnes, E., Donoway, E., Pavlick, E., Rodola, E., Lam, E., Chu, E., Tang, E., Erdem, E., Chang, E., Chi, E. A., Dyer, E., Jerzak, E., Kim, E., Manyasi, E. E., Zheltonozhskii, E., Xia, F., Siar, F., Martínez-Plumed, F., Happé, F., Chollet, F., Rong, F., Mishra, G., Winata, G. I., de Melo, G., Kruszewski, G., Parascandolo, G., Mariani, G., Wang, G., Jaimovitch-López, G., Betz, G., Gur-Ari, G., Galijasevic, H., Kim, H., Rashkin, H., Hajishirzi, H., Mehta, H., Bogar, H., Shevlin, H., Schütze, H., Yakura, H., Zhang, H., Wong, H. M., Ng, I., Noble, I., Jumelet, J., Geissinger, J., Kernion, J., Hilton, J., Lee, J., Fisac, J. F., Simon, J. B., Koppel, J., Zheng, J., Zou, J., Kocoń, J., Thompson, J., Wingfield, J., Kaplan, J., Radom, J., Sohl-Dickstein, J., Phang, J., Wei, J., Yosinski, J., Novikova, J., Bosscher, J., Marsh, J., Kim, J., Taal, J., Engel, J., Alabi, J., Xu, J., Song, J., Tang, J., Waweru, J., Burden, J., Miller, J., Balis, J. U., Batchelder, J., Berant, J., Frohberg, J., Rozen, J., Hernandez-Orallo, J., Boudeman, J., Guerr, J., Jones, J., Tenenbaum, J. B., Rule, J. S., Chua, J., Kanclerz, K., Livescu, K., Krauth, K., Gopalakrishnan, K., Ignatyeva, K., Markert, K., Dhole, K. D., Gimpel, K., Omondi, K., Mathewson, K., Chiafullo, K., Shkaruta, K., Shridhar, K., McDonell, K., Richardson, K., Reynolds, L., Gao, L., Zhang, L., Dugan, L., Qin, L., Contreras-Ochando, L., Morency, L.-P., Moschella, L., Lam, L., Noble, L., Schmidt, L., He, L., Colón, L. O., Metz, L., Şenel, L. K., Bosma, M., Sap, M., ter Hoeve, M., Farooqi, M., Faruqui, M., Mazeika, M., Baturan, M., Marelli, M., Maru, M., Quintana, M. J. R., Tolkiehn, M., Giulianelli, M., Lewis, M., Potthast, M., Leavitt, M. L., Hagen, M., Schubert, M., Baitemirova, M. O., Arnaud, M., McElrath, M., Yee, M. A., Cohen, M., Gu, M., Ivanitskiy, M., Starritt, M., Strube, M., Swedrowski, M., Bevilacqua, M., Yasunaga, M., Kale, M., Cain, M., Xu, M., Suzgun, M., Walker, M., Tiwari, M., Bansal, M., Aminnaseri, M., Geva, M., Gheini, M., T, M. V., Peng, N., Chi, N. A., Lee, N., Krakover, N. G.-A., Cameron, N., Roberts, N., Doiron, N., Martinez, N., Nangia, N., Deckers, N., Muennighoff, N., Keskar, N. S., Iyer, N. S., Constant, N., Fiedel, N., Wen, N., Zhang, O., Agha, O., Elbaghdadi, O., Levy, O., Evans, O., Casares, P. A. M., Doshi, P., Fung, P., Liang, P. P., Vicol, P., Alipoormolabashi, P., Liao, P., Liang, P., Chang, P., Eckersley, P., Htut, P. M., Hwang, P., Miłkowski, P., Patil, P., Pezeshkpour, P., Oli, P., Mei, Q., Lyu, Q., Chen, Q., Banjade, R., Rudolph, R. E., Gabriel, R., Habacker, R., Risco, R., Millière, R., Garg, R., Barnes, R., Saurous, R. A., Arakawa, R., Raymaekers, R., Frank, R., Sikand, R., Novak, R., Sitelew, R., LeBras, R., Liu, R., Jacobs, R., Zhang, R., Salakhutdinov, R., Chi, R., Lee, R., Stovall, R., Teehan, R., Yang, R., Singh, S., Mohammad, S. M., Anand, S., Dillavou, S., Shleifer, S., Wiseman, S., Gruetter, S., Bowman, S. R., Schoenholz, S. S., Han, S., Kwatra, S., Rous, S. A., Ghazarian, S., Ghosh, S., Casey, S., Bischoff, S., Gehrmann, S., Schuster, S., Sadeghi, S., Hamdan, S., Zhou, S., Srivastava, S., Shi, S., Singh, S., Asaadi, S., Gu, S. S., Pachchigar, S., Toshniwal, S., Upadhyay, S., Shyamolima, Debnath, Shakeri, S., Thormeyer, S., Melzi, S., Reddy, S., Makini, S. P., Lee, S.-H., Torene, S., Hatwar, S., Dehaene, S., Divic, S., Ermon, S., Biderman, S., Lin, S., Prasad, S., Piantadosi, S. T., Shieber, S. M., Misherghi, S., Kiritchenko, S., Mishra, S., Linzen, T., Schuster, T., Li, T., Yu, T., Ali, T., Hashimoto, T., Wu, T.-L., Desbordes, T., Rothschild, T., Phan, T., Wang, T., Nkinyili, T., Schick, T., Kornev, T., Tunduny, T., Gersten-

berg, T., Chang, T., Neeraj, T., Khot, T., Shultz, T., Shaham, U., Misra, V., Demberg, V., Nyamai, V., Raunak, V., Ramasesh, V., Prabhu, V. U., Padmakumar, V., Srikumar, V., Fedus, W., Saunders, W., Zhang, W., Vossen, W., Ren, X., Tong, X., Zhao, X., Wu, X., Shen, X., Yaghoobzadeh, Y., Lakretz, Y., Song, Y., Bahri, Y., Choi, Y., Yang, Y., Hao, Y., Chen, Y., Belinkov, Y., Hou, Y., Hou, Y., Bai, Y., Seid, Z., Zhao, Z., Wang, Z., Wang, Z. J., Wang, Z., and Wu, Z. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023. URL https://arxiv.org/abs/2206.04615.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL https://arxiv.org/abs/1804.07461.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.