AI Readiness: A Reusability Study of Popular AI Algorithms

Rob Quick
Indiana University
rquick@iu.edu

Mohith Surya Kiran Kasula
Indiana University
mkasula@iu.edu

Abstract

The FAIR Data Principles of findability, accessibility, interoperability, and reusability provide a roadmap to reusing data analysis findings and reproducibility of AI-based data analysis. However, the work done during this research project has identified an issue that impacts AI reproducibility before code and data interoperability can be considered. Namely, code reusability when attempting to recreate the hardware and system-level software or the "runtime environment." While attempting to determine the metadata needed to FAIRly couple datasets with AI algorithms, the research team determined that the problem of recreating the runtime environment of published state-of-the-art algorithms from the website Papers with Code provided a hurdle that must be overcome before automated data-algorithm coupling can be considered. While containerization solutions such as Docker or Singularity are created to address the issue of inconsistent runtime environments, few AI algorithm developers have embraced publishing containers alongside their AI codes, opting for documenting software dependencies, which only tell part of the runtime story. Additionally, containers are software, and many issues affecting the recreation of runtime environments can also affect orchestrated container solutions. This work describes the process employed to survey 75 openly available AI algorithms, as recorded by Papers with Code, spanning the machine learning areas of computer vision, audio analysis, and natural language processing. It also makes a case that merely publishing the algorithm software repository and datasets used to benchmark the accuracy of the analysis is not enough to enable the reproducibility of results or reuse of AI algorithms. Finally, it identifies the gap in runtime environment reusability between code repositories like GitHub and commercial services like Hugging Face to focus future work. It proposes providing solutions like a container library, enhanced documentation, and other methods to allow reproducible and reusable research and a roadmap for continuing toward a review of enhancing AI-ready data.

Keywords: Reproducibility, Artificial Intelligence, Machine Learning, FAIR AI, Reusability

1. Introduction

The AI reproducibility crisis is well known. Hutson (2018) writes, "AI researchers have found it difficult to reproduce many key results, and that is leading to a new conscientiousness about research methods and publication protocols." While this article focuses on publishing code, the same can be said about the environment in which the code is developed. Hutson quotes Peter Henderson, "...everyone kind of cooked up their own environments for their experiments, and that made it hard to compare results across papers..." O. Gundersen et al. (2018) identify this problem by stating, "...problems due to non-determinism in standard benchmark environments and variance intrinsic to AI methods require proper experimental techniques and reporting procedures. Acknowledging these difficulties, computational research should be documented properly so that the experiments and results are clearly described." While Haibe-Kains et al. (2020) goes a step further stating, "...the lack of details of the methods and algorithm code undermines its scientific value." All three publications referenced above conclude that things are slowly getting better, but work needs to continue to incentivize, document, and standardize publication methods before reproducibility becomes common.

Consider the persona of a researcher competent at using software applications in a cloud or high-performance computing environment, but does not have the skills or time available to write in-depth software applications. This researcher, while heavily vested in data-centric computing and modeling to do data analysis, may not be willing to start from scratch. Hence, they will look for existing artificial intelligence (AI) or machine learning (ML) code bases to adopt into their research environment. This researcher may be willing to do, or have access for support of some system-level porting, such as installing the necessary software dependencies and configuring the software to find the required libraries and paths to access data, but will not modify the code base directly. This is a common barrier to HPC adoption for many researchers, as noted by Mehringer et al. (2024).

In their paper, O. Gundersen (2021) states, "...reproducing an experiment by executing the same code and data on a different computer with different ancillary software introduces variability." This environmental variability when publishing software to code repositories like GitHub becomes obvious during attempts to reuse machine learning code. This is emphasized by Leventi-Peetz and Östreich (2020) when stating that, "...hidden bugs in the source code can lead to different outcomes in dependence of linked libraries and different execution environments." However, both were concerned with the reproducibility of published results and not taking code published by a developer and reusing it to build new models or analyze their datasets. It is essential to state that this paper will look at reusability rather than reproducibility, which, while similar, are not the same concept. We will not attempt to reproduce results from the publications surveyed but rather reuse the software in a generic cloud environment.

Several well-recognized efforts have been proposed to address the findability, accessibility, interoperability, and reusability (FAIRness) of data and software. The FAIR principles are defined in the seminal 2016 paper by Wilkinson et al. (2016) titled The FAIR Guiding Principles for Scientific Data Management and Stewardship. These guiding principles have been customized and adopted for software publishing by Lamprecht et al. (2020) and for instrumentation by Plomp (2020). Lamprecht et al. (2020) in particular mentions that "...software is data and software is not data." During this project, publishing experimental algoritm results on Papers with Code and publishing software to GitHub were seen as acceptably addressing the findability and accessibility components for the algorithms in question. We focused our experimental efforts on the interoperability and reusability principles. Again Lamprecht et al. (2020) assert "…interoperability has proven to be the most challenging principle, calling for particular attention in future discussions." Concluding that easing the substantial burden on the software developer will be a major step in addressing interoperability and reusability concerns. Here, we attempt to further that discussion by looking at where problems are encountered when reusing published code on open academic systems and resources.

In many cases, container technologies like Docker (Merkel (2014)), Singularity (Kurtzer et al. (2017)), and Apptainer (Dykstra (2024)) can be used to migrate applications by packaging dependencies, libraries, configuration files, and binaries. This adds another level of complexity to the publishing process and is often not the key concern of AI algorithm developers. Additionally, Watada et al. (2019) states, "...challenges like lack of cross-platform support and container portability limitations…" affect the usefulness of containers in cases of reuse.

By setting up virtual machine instances on the open academic cloud resource Jetstream2 described by Hancock et al. (2021) and attempting to recreate the runtime environment that allowed us to build new models or analyze new data, we were able to test reusability. We systematically attempted to reuse 75 published AI algorithms. When a prebuilt model was supplied we did not attempt to rebuild the model, if no model was published we started from building that model. These algorithms comprised three crucial machine learning categories: computer vision, audio analysis, and natural language processing. To ensure that we addressed the diversity of methods used in machine learning techniques, we divided each category into five (5) subcategories. These subcategories are recorded in the methods section below.

2. Methods

The first task was to identify target AI algorithms. The MetaAI Reserch-funded Papers with Code repository's mission is to "...create a free and open resource with Machine Learning papers, code, datasets, methods, and evaluation tables." MetaAI-Research (2024) The ease of accessing published papers, software repositories, and datasets through a single interface made this the obvious choice to gather our initial target algorithms. This published paper repository divided state-of-the-art ML algorithms into domains and subdomains. To ensure sampling diversity, we identified three domains: computer vision, audio analysis, and natural language processing. The bulleted list below identifies the domains and subdomains chosen based on the availability of multiple published results.

- Computer Vision
  - Image Classification
  - Denoising
  - Semantic Segmentation
  - Object Detection
  - Image Generation
- Audio Analysis
  - Voice Conversion
  - Language Identification
  - Music Generation
  - Audio Classification
  - Sound Event Detection
- Natural Language Processing
  - Language Modeling
  - Machine Translation
  - Sentiment Analysis
  - Question Answering
  - Text Generation

To ensure reuse was possible by a competent, but not expert, technical researcher, we limited the amount of modifications we would make to an algorithm in attempting to set up and execute the runtime environment to reuse the identified algorithms. We defined the necessary basic steps and limited our installation and configuration actions accordingly. These limitations included (1) we would not seek documentation beyond what was referred to in the README file, (2) we would not modify the downloaded code base, (3) we would not reconfigure hardware beyond what was identified in the README file, and (4) we would not modify data beyond what was identified in the README file. If the README file is linked to outside instructions, we did attempt to follow the instructions at those links, but we did not search for any additional documentation during the installation and configuration process. Likewise, if the links did not point at specific instructions (such as a project homepage), we did not attempt to search the site for additional instructions.

This workflow consisted of using the previously identified computer vision, audio analysis, and natural language processing algorithms and 1) creating a Linux-based virtual image on the Jetstream2 cloud hosted by Indiana University, 2) Downloading the AI algorithm code from its published repository, 3) installing the environment as directed within the README file of the repository, 4) attempting to execute sample training and/or analysis operations, 5) if step 4 was successful we recorded the metadata needed by the code executable for later consideration. This metadata was not used for this project but was recorded as a starting point for future work.

2.1. Machine and OS-Level Step Up

During the virtual machine instance creation process on Jetsteam2, we allocated 4 GPUs and 60 GB of disk space for each virtual image. This allowed us to utilize the GPU and CPU capabilities needed for most image analysis, audio analysis, and natural language processing implementations. The NSF-funded Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support, better known as ACCESS-CI (2024), handled the resource allocation process. The Exosphere Openstack cloud interface was used to create consistent and straightforward virtualized operating systems. Unless directed otherwise within the algorithm instructions, we used Ubuntu 22.04 as the base operating system.

Anaconda (2023) Miniconda was installed on each instance to allow easy installation of prepackaged and preconfigured software versions. This was coupled with the command line preferred installer program (pip) to obtain and install Python packages where necessary.

2.2. Code Download

All identified algorithms had GitHub repositories published on Papers with Code, which simplified the initial retrieval of the code base. The `git clone` command was used to download the contents of the repositories and ensure we were working with the exact code base intended by the developers upon paper publication.

2.3. Runtime Environment Setup

With a few minor exceptions, we limited our installation interactions to doing only steps identified in the developer-supplied documentation (README file). As stated earlier, we leveraged Miniconda virtual environments to ensure that the environmental updates were encapsulized and that the run-time environment was fully implemented as instructed in the developer documentation. We did not directly edit any Python code beyond the documented configuration provided by the code developers. However, we installed any missing Python modules identified during execution and updated command line operations aliases that were inconsistent between the targeted code base and environment.

If software packages identified by the README file or recorded in requirements.txt were not available we used the closest available version greater than the version identified. This method allowed us to check on versions that were similar if not exact and emulates the expected behavior of the researcher persona identified in the introduction of this paper.

We also downloaded and staged any necessary datasets identified in the README file. Depending on the size of the dataset this was done on the local VM disk if the dataset was less than 50GB or a mounted volume if the dataset was larger than 50GB as is consistent with the storage mechanisms on Jetstream2.

2.4. Build and Analysis Attempted

Model training was attempted on all code bases that included training methods but did not include pre-built models. If a model was built successfully, an analysis was attempted on that model. However, some targeted algorithms were packaged with pre-trained models and not with the training method. If this was the case, we attempted an analysis using the pre-trained model and did not build the model locally.

The targeted code was executed as directed in the README file with the necessary adjustments to the command line arguments. This was done to identify paths to data, destination paths, available CPUs or GPUs, direct analysis input, and other environmentally relevant information. If errors were encountered, all relevant output was captured and recorded.

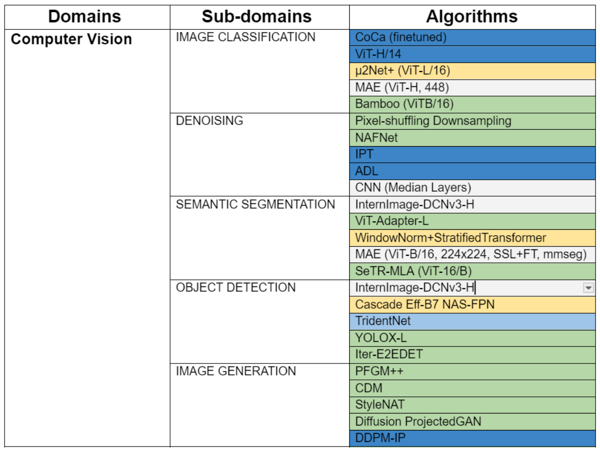
2.5. Record Input Parameters, Metadata Requirements, and Archiving Environments

If the code execution was successful, we recorded the command line parameters fed into the execution, including environment, data, and application parameters. This information, along with the successful step completion, was stored in a spreadsheet for results analysis.

After each algorithm analysis, the Jetstream2 virtual machine instances were preserved by creating snapshots for reuse in future work rather than deleting them. This will allow us to revisit the results during future research.

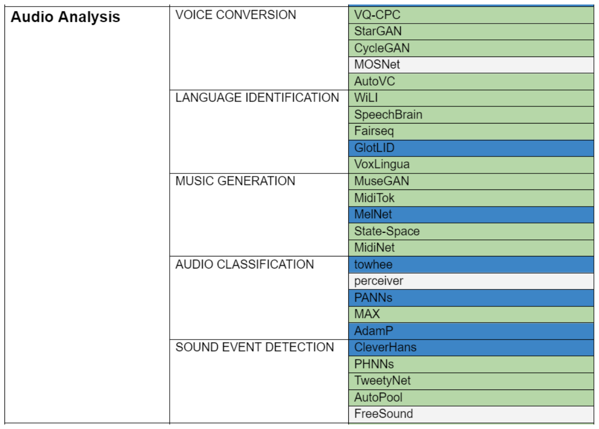
3. Results

The full details of the data collection can be seen in the data collection spreadsheet compiled by Quick and Kasula (2024). A condensed version of this data, along with the final status of each test, can be seen in figures 1-3 with a color key in figure 4.



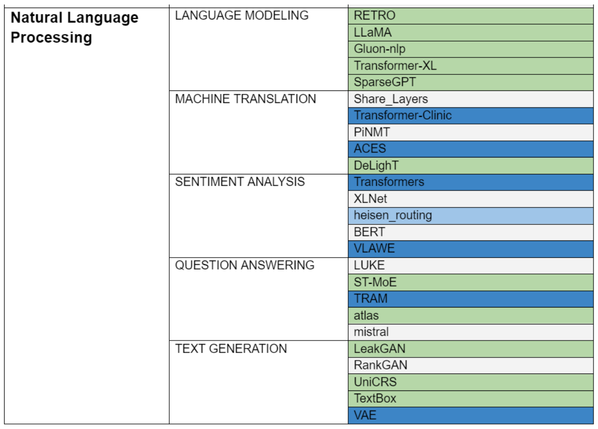
Domains	Sub-domains	Algorithms
Computer Vision	IMAGE CLASSIFICATION	🟦 CoCa (finetuned)
		🟦 ViT-H/14
		🟨 μ2Net+ (ViT-L/16)
		⬜ MAE (ViT-H, 448)
		🟩 Bamboo (ViTB/16)
	DENOISING	🟩 Pixel-shuffling Downsampling
		🟩 NAFNet
		🟦 IPT
		🟦 ADL
		⬜ CNN (Median Layers)
	SEMANTIC SEGMENTATION	⬜ InternImage-DCNv3-H
		🟩 ViT-Adapter-L
		🟨 WindowNorm+StratifiedTransformer
		⬜ MAE (ViT-B/16, 224x224, SSL+FT, mmseg)
		🟩 SeTR-MLA (ViT-16/B)
	OBJECT DETECTION	⬜ InternImage-DCNv3-H
		🟨 Cascade Eff-B7 NAS-FPN
		🔷 TridentNet
		🟩 YOLOX-L
		🟩 Iter-E2EDET
	IMAGE GENERATION	🟩 PFGM++
		🟩 CDM
		🟩 StyleNAT
		🟩 Diffusion ProjectedGAN
		🟦 DDPM-IP

Figure 1. Color coded results of Computer Vision domain in the reusability study.



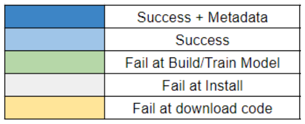
Audio Analysis	VOICE CONVERSION	🟩 VQ-CPC
		🟩 StarGAN
		🟩 CycleGAN
		⬜ MOSNet
		🟩 AutoVC
	LANGUAGE IDENTIFICATION	🟩 WiLI
		🟩 SpeechBrain
		🟩 Fairseq
		🟦 GlotLID
		🟩 VoxLingua
	MUSIC GENERATION	🟩 MuseGAN
		🟩 MidiTok
		🟦 MelNet
		🟩 State-Space
		🟩 MidiNet
	AUDIO CLASSIFICATION	🟦 towhee
		⬜ perceiver
		🟦 PANNs
		🟩 MAX
		🟦 AdamP
	SOUND EVENT DETECTION	🟦 CleverHans
		🟩 PHNNs
		🟩 TweetyNet
		🟩 AutoPool
		⬜ FreeSound

Figure 2. Color coded results of Audio Analysis domain in the reusability study.



Natural Language Processing	LANGUAGE MODELING	🟩	RETRO
		🟩	LLaMA
		🟩	Gluon-nlp
		🟩	Transformer-XL
		🟩	SparseGPT
	MACHINE TRANSLATION	⬜	Share_Layers
		🟦	Transformer-Clinic
		⬜	PiNMT
		🟦	ACES
		🟩	DeLighT
	SENTIMENT ANALYSIS	🟦	Transformers
		⬜	XLNet
		▨	heisen_routing
		⬜	BERT
		🟦	VLAWE
	QUESTION ANSWERING	⬜	LUKE
		🟩	ST-MoE
		🟦	TRAM
		🟩	atlas
		⬜	mistral
	TEXT GENERATION	🟩	LeakGAN
		⬜	RankGAN
		🟩	UniCRS
		🟩	TextBox
		🟦	VAE

Figure 3. Color coded results of Natural Language Processing domain in the reusability study.



🟦	Success + Metadata
▨	Success
🟩	Fail at Build/Train Model
⬜	Fail at Install
🟨	Fail at download code

Figure 4. Legend for color coded results

3.1. Machine and OS-Level Step-Up Results

All VMs were created successfully on Jetstream2. This is less a reflection on this research, but rather a statement that the hardware and operating system environment were accessible, stable, and worked as expected. This led to a 100% success rate for creating the core environment needed to begin the download of code.

3.2. Code Download Results

Out of the 75 algorithms examined, only 3 did not have contents that allowed us to continue onto the environment installation step. One (mu2net) failed due to the accessibility of the repository. One (Cascade) did not contain any executable software, and while it was downloadable only contained the text and images from the published paper. One (WindowNorm) stated, “The code will be released soon” in the README file.

3.3. Runtime Environment Setup Results

15 of the 75 algorithms repositories led to errors before or during the installation. Failures during installation point to incompatibilities between the software libraries used to develop the algorithm and the operating system. The cause of these failures can be categorized as one of the following:

- Code download failed (see Section 3.2).
- Unsupported software errors: The software-defined in the README file was not supported by the virtual machine’s operating system or hardware.
- Unsupported compiler errors: The compilation was not possible due to compiler-level errors. The compiler version was rarely reported in the README file, and unless defined, we did not attempt alternate compilers.
- Version compatibility errors: The software-defined in the README had version dependency clashes, causing errors during package retrieval and installation.

3.4. Build and Analysis Results

56 of the 75 algorithms repositories led to errors before or during the code execution phase. Failures during this stage reflect some code-level incompatibilities with the operating system or hardware. The cases for these errors are summarized as:

- Download or environment installation failures (see Section 3.2 and 3.3)
- Module import error or modules not found: This happened most if the module method names had been updated and the code had not been updated. Changing this would have taken code-level adjustments, and the experiment methods did not allow this level of modification.
- CUDA errors: CUDA driver incompatibility was one of the most common errors. This included PyTorch and Tensorflow incompatibility with CUDA versions, missing CUDA libraries, and other CUDA runtime errors.
- Slurm requirement not met: While uncommon some algorithms required Slurm batch management. Since the tests occurred on single nodes we installed Slurm for single nodes using the instructions provided by Santana (2024).
- Failure to download checkpoint files. These externally (to GitHub) supplied files may have moved or been removed by the developers.
- Failure to locate and access files needed for data analysis.
- Failure to recognize GPUs: GPUs were not found or recognized as available by the model build or data analysis software. This error is closely related to the CUDA errors mentioned above.
- Dataset could not be downloaded: Datasets were not found, were not accessible, or were corrupted.

3.5. Record Input Parameters, Metadata Requirements, and Archive Environment Results

One of the tested algorithms completed without visible error, but did not return any results. This is most likely due to a silent error, but since no error code or message were returned we were unable to determine the exact status. Another of the tested algorithms predefined all inputs so while results were returned there was no user level input of data or parameters necessary. These two tests account for the difference in the code executed and metadata recorded workflow steps.

The combined success rates for each workflow step described in the Methods section are shown in the charts below (figures 5-9).

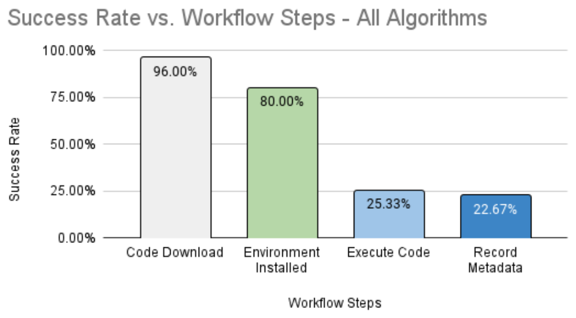


Figure 5. Success Rate vs. Workflow Steps - All Algorithms Combined.

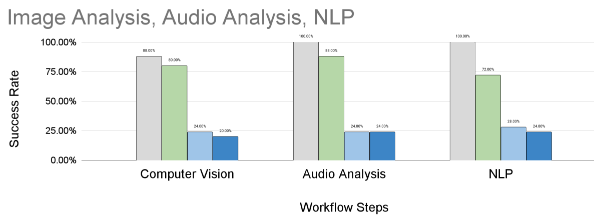


Figure 6. Success Rate vs. Workflow Steps: Comparative chart of Image Analysis, Audio Analysis, NLP.

The results show that code findability and accessibility of the surveyed algorithms had a high level of success, achieving a 96% rate. The documented runtime environment was able to be recreated at an 80% rate. However, a large drop off was experience when trying to execute or build models with the code. Only 19 of the 75 or 25.3% of the identified algorithms were able to successfully complete the installation and execution of the AI algorithm as described in the code repository documented installation procedures.

Only 22.7% of the recorded successes allow the successful collection of the necessary metadata to consider automated reusability methods. This metadata consisted of three general categories of data.

- Environmental configuration data (ie. data directory, output directory, number of GPUs)
- Data formatting data (input file format, normalization technique)
- Algorithm parameters (diffusion steps, model selection, scale)

All 75 Jetstream2 VM instances created during this research have been preserved by snapshotting the VM images for future work.

4. Conclusion

At the start of this research project it was envisioned as a dataset and AI algorithm interoperability study. However, due to the vast diversity of computing environments used to develop AI algorithms and the lack of motivation for developers to fully document and publish their runtime environment details, the ability to continue the research, as originally defined, stalled. What was originally seen as a study of dataset interoperability became a AI algorithm reusability study.

Our initial findings are consistent with other studies in the area. O. E. Gundersen and Kjensmo (2018) concluded that less than a third of AI research is reproducible. In the same article, it was reported that in a survey of 400 AI papers at conferences, only 6% included code, 30% included test data, and 54% included only pseudo-code. Gunderson's and Kjensmo's research did not take the hands-on approach of trying to install the code that was taken in this research. They merely checked papers for code, data, and pseudo-code references.

This survey of 75 highly-ranked computer vision, audio analysis, and natural language processing algorithms identified the difficulties in reproducing and reusing AI software. While this problem is being addressed in the commercial arena, there is currently little effort to address it in the academic space. The reproducibility of runtime environments is a long-standing problem and has been addressed on a technical level using containerization software such as Docker or Singularity. However, this is a software-based solution and is subject to the same versioning issues as the ones encountered in this experiment. Containers can and do get stale with time and may not provide a full solution to the runtime environment issues we have experienced.

Additionally, this survey points out the lack of motivation for AI algorithm developers to go beyond adding code to public repositories. Ideally, developers would publish containers along with publishing data and software. This additional step would increase, but not fully address, some portion of the reproducibility, reuse, and adoption problems encountered during this research project. However, it would add a burden to the algorithm developers. Another potential solution is to use standardized environments like those provided by HuggingFace or Google Colab. These options come with a paywall if you use more than a trivial amount of resources, with GPUs being especially expensive.

In the academic space, developing AI-friendly interfaces like a Science Gateway (Apache Airavata, Tapis, OneSciencePlace) or an HPC-powered web interface (Open OnDemand) could help increase reusability, again adding to the effort needed from the algorithm developer and by the interface administrator.

Until standard publishing of the data, software, and environment becomes common practice, this reuse crisis will continue. This digital artifact publication can be encouraged by incentivizing the publication of runtime environments, much as recent trends have incentivized data and code publication. In the US, this can be seen in the recent Nelson memo, which requires the public availability of federally funded research Nelson (2022) and focuses on digital products of research. The research team can and will evangelize this solution, but significant changes to the research AI community's approach to reusability is needed to evolve the incentivization process. However, while we can make our voices heard, we hold little influence in publication policy, and a more technical solution may be a better approach.

Another future solution may include using AI methods to document, publish, and ultimately recreate the necessary connections between algorithm, data, and runtime environments to enhance reusability. While this seems like an ideal inward-facing solution (using AI to make better AI), this consideration is beyond the scope and capability of this small research team. At the time of this paper composition, we could not identify any similar active research specifically in this area using AI. However, since this solution is so appealing, we will watch for activity on this front.

An additional technical solution would include hosting and maintaining container libraries that provide compatible runtime environments coupled with matching AI algorithms. This approach, discussed below in the Future Work section, is the planned continuation of this research project.

5. Future Work

The research team still plans to conduct the originally envisioned dataset interoperability study. We will use the results of this study to identify a set of algorithms that do not create time-consuming reusability issues. Starting from that subset of algorithms, we plan to provide a containerized public library that will simplify the reuse problems encountered in this study.

With that container library in hand, we will study training and analysis tasks with benchmarking datasets. This will allow us to identify common metadata needs for the AI sub-domains of computer vision, audio analysis, and natural language processing. After collecting these metadata requirements, we will engage with national and global organizations dedicated to reproducibility and open sharing to recommend and vet common AI metadata schemas via open community review and feedback.

The final goal of this research is to make a positive and meaningful step toward AI-readiness in research environments by providing researchers with the tools they need to easily reuse existing algorithms and data to further their research visions.

6. Acknowledgements

This work was done under the tutelage of Dr. Saptarshi Purkayasta at the Purkayasta Lab for Health Innovation.

References

ACCESS-CI. (2024). *Advanceing innovation with access*. https://access-ci.org/about/ (accessed: 06.15.2024).

Anaconda. (2023). *Miniconda*. https://docs.conda.io/en/latest/ (accessed: 06.14.2024).

Dykstra, D. (2024). Apptainer without setuid. *EPJ Web of Conferences*, (295), 07005.

Gundersen, O., Gil, Y., & Aha, D. (2018). On reproducible ai: Towards reproducible research, open science, and digital scholarship in ai publications. *AI Magazine*, (39(3)), 56–68.

Gundersen, O. E., & Kjensmo, S. (2018). State of the art: Reproducibility in artificial intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, (Vol. 32, No. 1).

Gundersen, O. (2021). The fundamental principles of reproducibility. *Philosophical Transactions of the Royal Society A*, (379(2197)), 20200210.

Haibe-Kains, B., Adam, G., Hosny, A., Khodakarami, F., Waldron, L., & Aerts, H. J. (2020). Transparency and reproducibility in artificial intelligence. *Nature*, (586(7829)), E14–E16.

Hancock, D., Fischer, J., Lowe, J., Snapp-Childs, W., Pierce, M., Marru, S., Coulter, J., Vaughn, M., Beck, B., Merchant, N., & Skidmore, E. (2021). Jetstream2: Accelerating cloud computing via jetstream. *Practice and Experience in Advanced Research Computing*, 1–8.

Hutson, M. (2018). *Artificial intelligence faces reproducibility crisis.* https://www.science.org/doi/full/10.1126/science.359.6377.725 (accessed: 06.14.2024).

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PIoS one*, (12(5)), e0177459.

Lamprecht, A., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., Van De Sandt, S., Ison, J., Martinez, P., & McQuilton, P. (2020). Towards fair principles for research software. *Data Science*, (3(1)), 37–59.

Leventi-Peetz, A., & Östreich, T. (2020). Deep learning reproducibility and explainable ai (xai). *arXiv preprint*, (2202.11452).

Mehringer, S., Thomas, M. P., Cahill, K., Dey, C., Joiner, D., Knepper, R., & Powell, J. H. (2024). Scaling hpc education. *Journal of Computational Science*, (15(1)).

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux j*, (12(5)), e0177459.

MetaAI-Research. (2024). *Our mission.* https://paperswithcode.com/about (accessed: 06.15.2024).

Nelson, A. (2022). *Ensuring free, immediate, and equitable access to federally funded research.* https://www.whitehouse.gov/wp-content/uploads/2022/08/08-2022-OSTP-Public-Access-Memo.pdf (accessed: 09.4.2024).

Plomp, E. (2020). Going digital: Persistent identifiers for research samples, resources and instruments. *Data Science Journal*, (19), 46–46.

Quick, R., & Kasula, M. (2024). *Workflow spreadsheet*. https://docs.google.com/spreadsheets/d/1reZ1Yp-22pBEpnHWUb2fSDSpeRU0VjlcGYOnZPlEUKg (accessed: 06.14.2024).

Santana, P. (2024). *Slurm single node installation on ubuntu 22.04.* https://medium.com/@pausantanapi2/slurm-single-node-installation-on-ubuntu-22-04-6479e70f7911 (accessed: 06.14.2024).

Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging trends, techniques and open issues of containerization: A review. *IEEE Access*, (7), 152443–152472.

Wilkinson, M., Dumontier, M., Aalbersberg, I., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J., da Silva Santos, L., Bourne, P., & Bouwman, B., J.and Mons. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific Data*, (3(1)), 1–9.