Navigating the Integration of Machine Learning into Domain Research

Bernie Boscoe
Department of Computer Science
Southern Oregon University
Ashland, Oregon 97520, USA

Tuan Do
Physics and Astronomy Department
University of California, Los Angeles
Los Angeles, CA 90095, USA

Noah Mogenson
Department of Computer Science
Southern Oregon University
Ashland, Oregon 97520, USA

***Abstract*—Increasingly, scientific research teams desire to incorporate machine learning into their existing computational workflows. Codebases must be augmented, and datasets must be prepared for domain-specific machine learning processes. Team members involved in software development and data maintenance, particularly research software engineers, can foster the design, implementation, and maintenance of infrastructures that allow for new methodologies in the pursuit of discovery. In this paper, we highlight some of the main challenges and offer assistance in planning and implementing machine learning projects for science.**

I. Introduction

Machine learning is becoming a useful tool for scientific research, enabling the discovery of patterns and insights in large and complex datasets. As a result, research teams seek to incorporate machine learning into their existing computational workflows. However, these integrations present numerous challenges, such as the need to augment codebases and prepare datasets for specific machine learning processes. In addition, team members involved in software development and data maintenance, such as research software engineers, must design, implement, and maintain infrastructures that allow for new methodologies in the pursuit of discovery. In this short paper, we aim to address some of the main challenges faced by research software engineers (RSEs) seeking to implement machine learning pipelines. While at a slower pace than industry adoption, machine learning (ML) techniques have been used in a variety of scientific domains, from physics and astronomy to biology and medicine, with applications ranging from image and signal processing to prediction and classification. For the RSE considering adoption techniques, robust implementation choices abound as tools from industry such as TensorFlow and Pytorch enter more stable phases, allowing for scientific exploration. We will discuss some of the challenges in integrating these new tools into existing scientific data infrastructures, pipelines, and software frameworks. The intended audience of this paper are people who understand ML concepts and are responsible for planning and implementing ML pipeline building and maintenance.

As a basis for the recommendations in this paper, we draw from our ongoing research project, Machine Learning in Astronomy, where we interview astrophysicists who are incorporating machine learning into their research. We have collected common threads from researchers that are adopting machine learning methods either to replace or augment more traditional tools in their field. We also utilize our own experience building machine learning pipelines in our Astrophysics Data Lab at UCLA.

Presented at USRSE2023, Chicago, IL, October 16-18, 2023.

II. Codes and Cadences

Machine Learning (ML) tool development in the technology industry has accelerated at a blistering pace. In 2023 alone we have seen stunning developments in Large Language Models (Chat-GPT) [13], and other improvements in edge computing [2] and face recognition [4]. In industry, rapid changes in code, models, and data are commonplace as part of the innovation process; however, in academic research, it can take considerably longer to achieve results. This is in part due to the importance of the scientific method: tracking and recording preliminary results, and using the appropriate metrics to determine successful results. So, too, do publication cycles affect project trajectories. For RSEs, these differences in scientific production cadences versus library updates can result in code incompatibilities and deprecation, as well as complications in the preservation of code that needs to be made available.

Preservation of processes and outputs are necessary to justify claims and replicate results [14]. One main difference between research using ML tools and research with traditional domain coding methods is that ML tools are increasingly imported from industry-originating code bases such as TensorFlow (Google) and PyTorch (Meta). While these code bases are both open source, their current user and contributor communities do not prioritize software preservation methods as needed in academic research such as documenting code changes, version control, and other related practices [6].

Software in scientific domains traditionally comprises software tools developed over the course of many years, has achieved trust, stability, and is often combined with code written from scratch for a problem at hand. Oftentimes these tools have been used for decades, and to introduce a previously unused, untested library inserts many unknowns into workflows, much like adding a new instrument to a study. The RSE might opt for stable libraries as opposed to developing bespoke code for a team. In this case, the RSE might be tasked to choose between TensorFlow and PyTorch; and while they have similar capabilities a code maintainer might find it easier to focus on just one. For RSEs, the additional work effort to maintain code used for ML might stand in stark contrast to traditional code base maintenance. For example, TensorFlow code (and hence models produced from it) written as recently as a few months prior might not run today, resulting in reproducibility and replicability issues [3].

Reasons for failed runs are often due to library incompatibilities, resulting in unresolvable conflicts among the many libraries necessary to run scripts. In this case an RSE must decide whether to use older, possibly deprecated versions of libraries, downgrade to earlier versions of libraries, or craft solutions of both to run scripts. An RSE might want to consider the goals of the research team with respect to the differences in reproducibility versus replicability, and reasonable expectations in preservation and maintenance of ML code. For example, does the team want to obtain the same results themselves using the same data, code, and hardware? Should other teams be able to accomplish similar results on different hardware and software? If code produces non-deterministic results, what are acceptable outputs? Documenting language versions, library versions, hardware specifications, OS, and software drivers to enable GPUs are examples of metadata necessary to produce verifiable results.

Using ML to explore large datasets in any domain necessarily requires the training and testing of many models; discarding the ineffective ones. ML APIs such as Keras for TensorFlow have developed over time to enable pipelines to be built and executed with a small amount of code, democratizing ML to be easily used by domain scientists yet creating issues for the RSE with respect to maintenance. For example, at present there are numerous ways to save training models, including saving only the model architectures or saving both architectures and weights. While current ML tools have architecture and model saving capabilities built into the APIs, the rapid evolution of the libraries themselves can cause issues with preservation [12]. For RSEs, staying abreast of current guidelines in research software development can help to inform best ML practices, for example utilizing code reviews and sprints, encouraging organizational tools like issue trackers and repositories that facilitate software reuse and code optimization [11, 10].

Another outcome of large companies such as Google and Meta influencing ML open source development is the choice of Python for APIs. While lower layers of code architectures are commonly written in C/C++, an RSE not well-versed in Python might need to learn the intricacies of open source libraries and, unfortunately, the 'dependency hells' they inevitably create. Researchers and other team members that write code might not take incompatibilities into consideration, as many researchers write code specifically for their own experiments. An RSE might create standardized pipelines for PIs, postdocs and students to follow, or even develop custom APIs.

In industry, the job title of Machine Learning Engineer is evolving to include pipeline developers and maintainers, sometimes called ML Ops specialists. The RSE will likely have to wear several hats as the software specialist. Putting models into production is a skill with a somewhat steep learning curve, and additional training is likely necessary for a software engineer, as the field is so new. The RSE should work with their team to determine the type of production the team seeks; it could be as basic as using fixed data and models to deploying a continuous integration pipeline. All pipelines require maintenance, and upkeep is likely due to rapid changes in ML libraries, as well as data drift if new data is being fed to the model.

The combination of the fragility of code libraries, the architecture that libraries are built on, and speed of development permeates all corners of scientific software development, and is at odds with the RSE's goal of stability and the consistency necessary to do good science [9]. Not only do the code, libraries, and ways to save models change, but the environments themselves are equally as fragile. Docker and Kubernetes containers, Jupyter notebooks, Conda or other environments must be kept alive or archived in such a way that the information can be obtained. Additionally, commercial platforms such as GitHub can also change over time, as corporate influences may change availability and cost of code and tools. Cost in terms of software is vague and difficult to define, but is important to consider, as it can have far-reaching implications regarding the practicality of research. There are always tradeoffs, and while cost can refer to monetary value, it can also refer to labor costs. Time spent learning a new (theoretically better) tool is time that could be spent using a familiar tool to perform another task. A typical cost-benefit analysis in ML determines whether to maintain a local instance of GPUs or to use cloud-based computing.

III. Machine learning-ready data

One potential pitfall facing RSEs is that their research group's data is not in a suitable form for machine learning pipelines [5]. We assume here that the RSE has sufficient data available to them, but the data has not been created or collected with the intention to be analyzed by downstream ML processes. A preliminary consideration is the file format(s) in which the data is stored. File formats might not be amenable to ML processes, for example, proprietary data formats from domain-based instruments are likely not able to be ingested by common ML tools without considerable transformation efforts. For newcomers to these tasks, a great deal of planning and preparation is needed to shape data to conform to ML tools. Whereas many datasets are created to be evaluated by a human end user, ML datasets must be read by machines as well as humans.

Two common forms of data for ML are images and tabular data. ML tools are most successful at ingesting well-defined data. For images, each image should be of a similar size and dimension to the others in the set, and for tabular data, each column, also known as a feature, represents a measurable property in some kind of standardized form. Data types for column features should be both internally and externally consistent. Internal consistency refers to each data point's characteristics being consistent with all the members in the column. External consistency refers to the appropriate scaling done on the columns, typically performed in a pipeline process, to prepare data for model training or during predictions.

Because ML tools read in data in multidimensional arrays (tensors), data records (rows) should all be of the same dimensions. The tradeoffs of ML data involve using the maximum amount of features and information that allow for a reasonable training compute time.

Over the years, a large number of datasets have been developed to train and test ML models, and many of these datasets are readily available; but topics are limited to applications important to industry and computer science, and may contain ethically problematic biases. Well known datasets include MNIST, a set of black and white images of hand-drawn numerals from 0-9, and ImageNet, containing over 14 million images (typically 256x256) and associated label information. One particular challenge for the RSE is the lack of generic, domain-based datasets for learners to use. Thus, while students may have learned basic ML techniques, they may not have experienced the challenges of transitioning to training and testing data from their respective domains. In domain sciences, ML datasets are typically not publicly available, and/or too specific in scope to allow for general tinkering.

PyTorch and TensorFlow libraries have been developed to optimally ingest specific types of data containing the smallest amount of information possible to yield results and minimize compute power, such as png and jpg for images, and csv files for tabular data. For image data, size in dimensions and pixels are considerations for both model intake and the amount of compute power needed to handle them. For tabular data, the number of columns, the data types stored within the columns, and the number of rows can all be serious factors in designing batching systems that consider limitations of RAM, training time, and compute power. Even for research teams who are comfortable with analyzing large datasets with long compute periods on, for instance, a cluster using a job scheduling software such as Slurm, might not be familiar with the resources required to train and evaluate an ML model.

Because of the way datasets may be separated into distinct subsets that are used to train models in ML, it is important to create well-defined datasets that adhere to scientific processes. If datasets are to be split, they must be divided into training, test, and possibly validation subsets and great care should be taken so that no contamination between sets takes place. If data is used to create a model then is subsequently used to validate the same model, this causes data leakage, resulting in overfitting as well as invalidating scientific inquiry. A data versioning system is needed to show which datasets were used to create specific models. In addition to deciding what data to build a model with, data might undergo changes within a pipeline, for example, feature engineering might take place. This involves selecting only important columns or combining columns of data into more statistically or categorically useful features. Versioning datasets in preprocessing and throughout model building is a critical component to ensure scientifically robust ML processes.

A. *Dataset versioning*

The granularity of data versioning is also an important design consideration. For example, how data versions are affected if data engineering takes place, new data is added to the set, or augmentation takes place should be documented. An RSE might opt to implement a data versioning system similar to software versioning, or like approaches taken within ML domains [8, 1]. As Gebru writes in [8], since there is no standardized way to describe datasets, careful consideration must be given to describe the origins of the data, particularly if data containing information about individuals is used. Documentation in the form of readme files external to the data, as well as internal data information located within files are both important for contextual evidence to the scientific inquiry.

Introducing dataset versioning and labeling to a research group unfamiliar with these practices can be time-consuming and difficult. Getting buy-in from PIs is necessary when changing traditional team workflows. We recommend an approach that releases a first version dataset as soon as possible, with improvements to follow in later iterations. This approach eschews planning phases, with an emphasis on implementation. Publications should align to specific datasets, and when possible, datasets should be stored in actual repositories such as Zenodo and Github, and when allowable, made publicly available. Zenodo DOIs can aid the process of versioning datasets relating to publications; however, ML datasets might be too large for current repositories, a vexing problem when trying to make data and code publicly available.

B. *Dataset analysis and storage platforms*

Big data storage and access can be an expensive issue for academic research groups. Additionally, having access to compute power is a separate and perhaps more challenging issue. Research groups may opt to use cloud computing, purchase their own equipment, or utilize existing institutional resources. For an RSE, how much compute power is actually required is challenging to determine. Factors include size and number of datasets, architectures used, and estimated length of training needed. Having team members with scant knowledge of virtual machine instance costs can prove to be an expensive endeavor.

Institutional cluster resources might be a safer choice, but may come with limited support and expensive pricing models, such as buying a node on the cluster in a yearly subscription. Some institutions and universities provide High Performance Computing (HPC) environments for ML, but these might be difficult to use, because the setup might not be optimized for machine learning. Finally, a research team might opt to purchase their own servers with GPU cards, avoiding mistakes of letting jobs run too long, but also likely requiring that the team itself is responsible for the upkeep, maintenance, server patching, and necessary environment configurations to enable research.

Presently, platforms like Amazon Web Services (AWS) allow for large, relatively inexpensive storage compared to the cost of server upkeep, but continuously running ML pipelines comes at a financial cost. In current climes, for a medium sized research group using ML, cloud solutions are likely too expensive, and a group-owned server, even with its drawbacks, is the likely most economic solution due to researchers needing to tinker, by running RAM-intensive jobs for long periods of time.

Some institutions might offer access to a cluster, for a fee or at no cost, with the added bonus that research groups have already had experience utilizing these resources. However, job allocation software such as Slurm might not work well with ML model building runs [7]. This is due in part to Slurm being general purpose. Users would have to implement such functionality themselves (which can be done, see https://github.com/y0ast/slurm-for-ml), or rely on additional tool integrations. However, this requires a deeper knowledge of the requirements of ML model training and the libraries being used. As noted previously, research teams without prior experience in the domain may not have such knowledge. Each additional software layer added to an ML pipeline will increase complexity, and lower-level tools like Slurm will have their own ways of interacting with hardware.

IV. Machine Learning Hardware

RSE's hardware expertise is important for machine learning because many software libraries are device-dependent. Support and documentation for device drivers for GPUs, required for machine learning, may be limited. Environments containing the correct GPU card drivers and libraries like CUDA are critical to ML processes, and may be difficult to configure. Likewise, libraries are needed to coordinate data being passed between the CPU and GPU(s), and code might need to be parallelized to maximize performance. CUDA and Python solutions are perhaps the easiest to implement, and the RSE might need to familiarize themselves with the abstraction hierarchy and where their expertise will match with the researchers' computational needs. Tools like NUMBA allow one to write their own CUDA kernels in Python, allowing for custom algorithms, or the research output can be satisfied using TensorFlow or Pytorch with no other customizations necessary. In requirements gathering, the RSE should attempt to pinpoint exact needs to determine the appropriate hardware configurations.

V. Getting Credit

RSE's can provide essential contributions to machine learning-enabled domain research. As an RSE's career path develops, it is wise for them to track accomplishments. Authoring publications can be a key metric associating software engineering work with domain research. RSEs can contribute their expertise and results in process papers and dataset creation papers, to name a few examples to receive credit for scientific work. Publications, such as the Journal for Open Source Software (JOSS), can document software package contributions for scientific aims. An RSE might opt to submit to a general ML journal, or specify their work for their respective domain; and care should be taken to follow domain norms of publication style and content. Publications are evolving over time to include more detailed information about data and code used in scientific research, and so opportunities for the RSE to showcase their work are increasing. Additionally, participating in publication activities as a reviewer will only serve the RSE gains a better understanding of the processes to publish. Lastly, organizations such as The United States Research Software Engineer Association (US-RSE) and The Software Sustainability Institute foster communities supporting the efforts of RSEs and their contributions to science via software development.

VI. Conclusion

Incorporating machine learning techniques and software into scientific research is an exciting, growing field. Initial transitions to ML software implementations will likely prove challenging, as domain researchers will have to alter their traditional software tooling methods, and learn new techniques unfamiliar to them. For RSEs, adopting new computational techniques can only serve to cement the necessity of technological expertise in collaborations with domain researchers. Our aim is to highlight possibilities for these special collaborations, in the spirit of furthering scientific exploration utilizing machine learning.

References

[1] Rabe Abdalkareem, Md Atique Reza Chowdhury, and Emad Shihab. *A Machine Learning Approach to Determine the Semantic Versioning Type of npm Packages Releases*. arXiv:2204.05929 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2204.05929. URL: http://arxiv.org/abs/2204.05929.

[2] Patricia Arroba et al. *Sustainable Edge Computing: Challenges and Future Directions*. arXiv:2304.04450 [cs]. Apr. 2023. DOI: 10.48550/arXiv.2304.04450. URL: http://arxiv.org/abs/2304.04450.

[3] Vishnu Banna et al. *An Experience Report on Machine Learning Reproducibility: Guidance for Practitioners and TensorFlow Model Garden Contributors*. arXiv:2107.00821 [cs]. July 2021. URL: http://arxiv.org/abs/2107.00821.

[4] Aaditya Bhat and Shrey Jain. *Face Recognition in the age of CLIP & Billion image datasets*. arXiv:2301.07315 [cs]. Jan. 2023. DOI: 10.48550/arXiv.2301.07315. URL: http://arxiv.org/abs/2301.07315.

[5] Bernie Boscoe, Tuan Do, and Evan Jones. "Elements of effective machine learning datasets in astronomy". In: *NeurIPS, Machine Learning for the Physical Sciences Workshop* (Dec. 2022). DOI: arXiv:2211.14401v2[astro-ph.IM].

[6] Stephen Crouch et al. “The Software Sustainability Institute: Changing Research Software Attitudes and Practices”. In: *Computing in Science Engineering* 15.6 (Nov. 2013). Conference Name: Computing in Science Engineering, pp. 74–80. ISSN: 1558-366X. DOI: 10.1109/MCSE.2013.133.

[7] Qiyang Ding et al. *Mirage: Towards Low-interruption Services on Batch GPU Clusters with Reinforcement Learning*. arXiv:2306.14086 [cs]. June 2023. DOI: 10.48550/arXiv.2306.14086. URL: http://arxiv.org/abs/2306.14086 (visited on 07/07/2023).

[8] Timnit Gebru et al. “Datasheets for Datasets”. In: *arXiv:1803.09010 [cs]* (Mar. 2018). arXiv: 1803.09010. URL: http://arxiv.org/abs/1803.09010.

[9] K. Hinsen. “Dealing With Software Collapse”. In: *Computing in Science Engineering* 21.3 (May 2019), pp. 104–108. ISSN: 1521-9615. DOI: 10.1109/MCSE.2019.2900945.

[10] Daniel S. Katz and Michelle Barker. *The Research Software Alliance (ReSA)*. en. other. News, Apr. 2023. DOI: 10.54900/zwm7q-vet94. URL: https://upstream.force11.org/the-research-software-alliance-resa.

[11] Daniel S. Katz et al. “Research Software Development & Management in Universities: Case Studies from Manchester’s RSDS Group, Illinois’ NCSA, and Notre Dame’s CRC”. In: *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*. arXiv:1903.00732 [cs]. May 2019, pp. 17–24. DOI: 10.1109/SE4Science.2019.00009. URL: http://arxiv.org/abs/1903.00732.

[12] *Migrate the SavedModel workflow — TensorFlow Core*. en. URL: https://www.tensorflow.org/guide/migrate/saved_model.

[13] OpenAI. *GPT-4 Technical Report*. arXiv:2303.08774 [cs]. Mar. 2023. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774.

[14] Roger D. Peng. “Reproducible Research in Computational Science”. en. In: *Science* 334.6060 (Dec. 2011), pp. 1226–1227. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1213847. URL: http://science.sciencemag.org/content/334/6060/1226.