Scientific/Technical/Management Plan

NASA expects over 100PB of data projected to be generated in its space science missions each year [42]. But since most of this data are generated on spacecraft, there is a bottleneck: it is expensive and sometimes infeasible to return the data to ground-based systems for processing. Given the NASA Strategic Plan's focus on the use of computationally expensive machine learning and artificial intelligence techniques [41], the use of lightweight and efficient data analysis techniques directly on the spacecraft is necessary. The importance of this is highlighted by the existence of research groups like JPL's MLIA [40] and the commercial AI solutions of companies like BAE [9], OCE Technology [44], and Edge Impulse [15].

However, in this setting, the machine learning software ecosystem is lacking. The specific hardware needs of spaceflight applications prevents the use of many common machine learning packages, and the balkanized complexity of the modern spaceflight hardware ecosystem makes deployment of machine learning models difficult due to poor/nonexistent documentation and discoverability of solutions.

Following both Recommendations 10 (novel computational techniques) and 11 (community education) of the 2024 SMD Strategy for Data Management and Computing for Groundbreaking Science [42], we propose a solution to this problem: the lightweight mlpack C++ machine learning library [21], already used in spaceflight applications [27, 28], is a promising and popular solution for low-resource machine learning deployments. mlpack uses the standard C++ compilation toolchain, and does not have onerous dependency requirements, which makes its integration in complex spaceflight applications trivial. We will extend the scope of low-resource devices that mlpack supports, enabling researchers to deploy mlpack to almost any commonly-used spaceflight computer. More importantly, we will focus on education and onboarding by developing tutorials, case studies, and seminars demonstrating the use of mlpack on resource-constrained devices, giving researchers an easy starting point for their particular task.

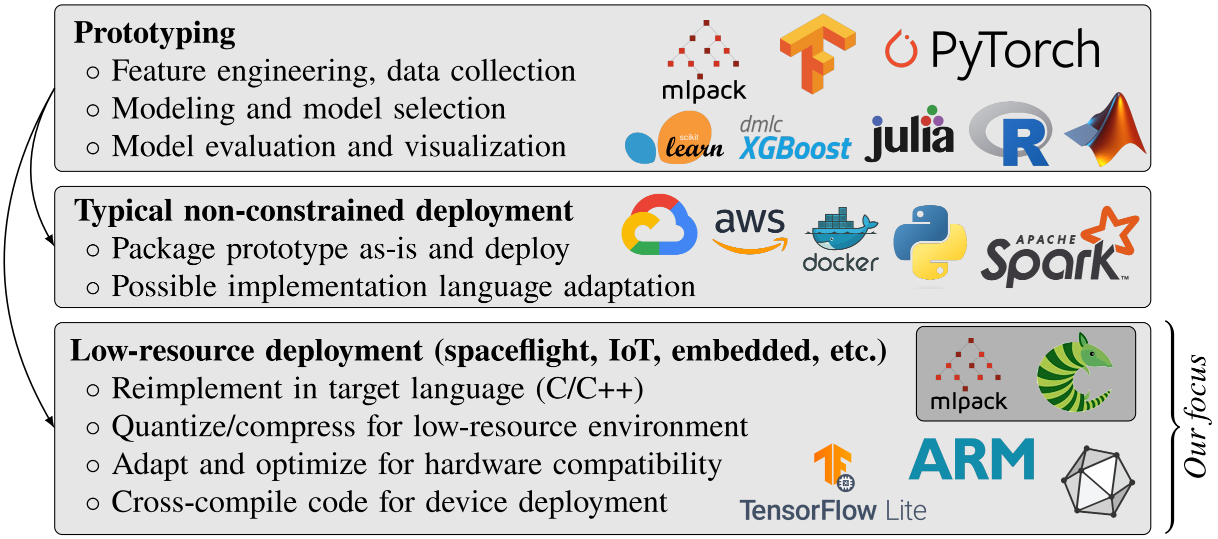
Prototyping
○ Feature engineering, data collection
○ Modeling and model selection
○ Model evaluation and visualization
mlpack
PyTorch
scikit learn
dmlc XGBoost
julia
R
Typical non-constrained deployment
○ Package prototype as-is and deploy
○ Possible implementation language adaptation
aws
docker
APACHE Spark
Low-resource deployment (spaceflight, IoT, embedded, etc.)
○ Reimplement in target language (C/C++)
○ Quantize/compress for low-resource environment
○ Adapt and optimize for hardware compatibility
○ Cross-compile code for device deployment
mlpack
ARM
TensorFlow Lite
Our focus

Figure 1: The data science workflow for low-resource and embedded devices (for example, spacecraft). When deploying to low-resource devices instead of typical targets such as cloud environments or Docker containers, the process is significantly more complex. Software that was used for prototyping often cannot be used in the deployment environment, necessitating time-consuming rewrites that are specific to the deployment hardware.

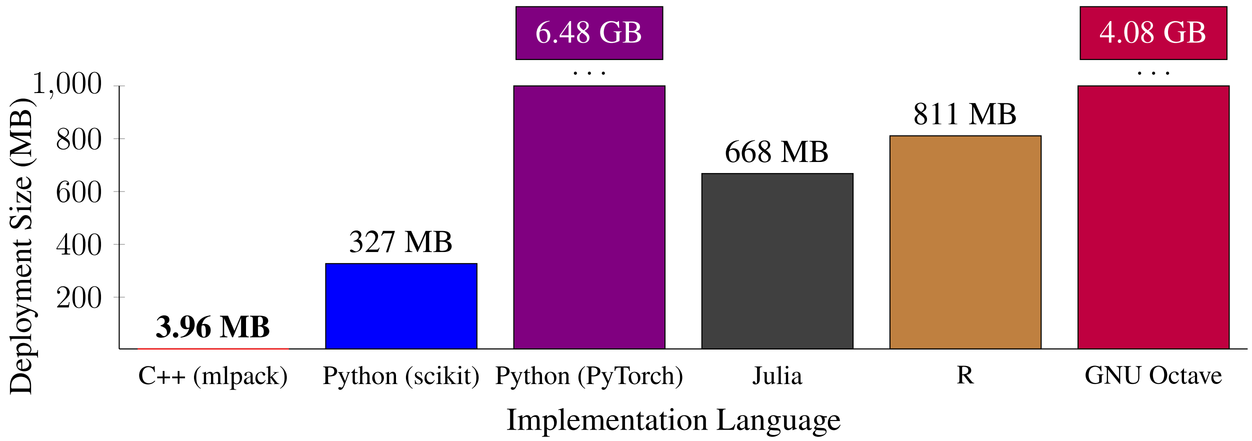

Figure 2: High-level solutions such as Python, R, Julia, and GNU Octave require heavyweight language runtimes. This causes the size of any machine learning solution to be orders of magnitude larger than a simple C++-based solution using mlpack. Here, we have measured the size of a Docker container containing only the code necessary to perform inference using a simple pretrained logistic regression model. The model size is 200KB and predicts the language of given input text using extracted features from that text.

Deployment difficulties for spaceflight machine learning

Researchers and practitioners have extremely limited time, and so it is very important to provide a simple set of tools that allow an idea to be quickly turned into production-ready code. This is a large part of what propelled Python to become the dominant language for machine learning: a focus on ease-of-use and adaptable examples [16, 52, 48]. But in the setting of resource-constrained machine learning, few simple tools exist, for several reasons:

- **Standard solutions are not suitable for spaceflight computers.** Typical Python-based solutions that scientists may be familiar with, such as PyTorch [46], TensorFlow [2], and scikit-learn [47] (to name only a few) are designed for prototyping and have too much overhead for low-resource deployments, due to the heavyweight Python runtime environment. Even popular non-Python solutions such as Julia [14], R [50], and MATLAB [30] (or the open-source equivalent, GNU Octave [23]) cannot be used, for the same reason. Figure 2 shows the total size of a machine learning deployment for a simple pretrained logistic regression model (of size 200KB). The Perseverance Mars Rover's computer has only 128MB RAM [51], and the James Webb Space Telescope uses a RAD750 with 44MB RAM [36]. As such, only the smaller mlpack application could be used for on-device machine learning.
- **On-device learning is often unsupported.** There are numerous solutions for neural network inference on low-resource devices; TFLite [59], TFLite-micro [22], ExecuTorch [49], the ONNX Runtime [11], and ArmNN [7] all allow neural network inference on a wide variety of devices. On-device learning is not possible with these toolkits; only inference is supported. But, in a spaceflight application, training or fine-tuning is often necessary or desired functionality [57, 62, 18, 12].

  Furthermore, standard classical machine learning algorithms such as decision trees, nearest neighbor search, and other common algorithms are not easily implementable in the neural network paradigm, and the tools above cannot be used for them. Even common preprocessing tools that may be used as a first step before a machine learning solution, such as PCA [56], or simple data scaling and normalization, are not available and if a workflow needs these components, they must be manually implemented.

- **Spaceflight processors are not standard hardware.** Space-rated computers often have non-standard ISAs (instruction set architectures), typically due to the need for radiation-hard equipment. The popular RAD750, which contains a PowerPC CPU, is used in over 100 satellites [10], and numerous other COTS spaceflight computing solutions contain PowerPC CPUs [29, 39]. The LEON line of SPARC CPUs [25] was recently selected for the SpaceLogistics MRV and MEP [17], and is used in the ESA's Galileo satellites [24]. Older space-grade CPUs like the Mongoose-V, which uses the MIPS ISA [58], are still in use on missions like TIMED [34] and New Horizons [35].

  But software availability for these architectures is limited; most languages' package managers do not support PowerPC, SPARC or MIPS. Most packages on the Python Package Index (PyPI) [1] do not provide packages for these architectures, making deployment of Python-based applications to these architectures especially cumbersome or even entirely infeasible.

- **Tooling for embedded hardware is opaque, difficult, and sometimes not open-source.** The most widely used hardware accelerators, Nvidia GPUs, are programmed with the closed-source CUDA language [43], and the majority of machine learning toolkits have robust support for these devices. Outside of the CUDA world, the software landscape is much more complicated. Other GPUs are often programmed with incompatible vendor-specific languages, such as AMD's HIP/ROCm [13], Intel's oneAPI [32], or Apple's Metal [4].

  Often, machine learning libraries do not have direct support for this wide array of technologies. For example, to use TensorFlow with HIP/ROCm, the primary option is the use of a custom Docker container that contains a fork of TensorFlow maintained by AMD [3]. Use outside of a Docker container may require custom compilation. Similarly, PyTorch does not support Intel oneAPI directly, but instead a separate Intel-maintained extension to PyTorch must be built and installed manually [31].

  The landscape for low-resource and embedded CPUs is similarly fragmented. For example, ARM's Cortex processors contain specific extensions for numerical computing, and these are supported via ARM's CMSIS library [6] and its associated neural network support library (CMSIS-NN) [8], which provides fast implementations of neural network operations. But, CMSIS-NN is not integrated with any common machine learning toolkit, and the ARM-provided documentation suggests a by-hand reimplementation of neural networks using low-level functions [5]. This necessitates a time-consuming and tedious manual conversion.

Easing spaceflight machine learning deployments with mlpack

These problems create a challenging landscape for the deployment of machine learning models on spacecraft. Currently, these problems are often solved by complex solutions specific to each individual use case, but this is not a good approach; it is much better to instead use simple, easy-to-use, composable tools that tap into researchers' existing knowledge. This is the UNIX philosophy [54, 53] that underlies so much of modern computing. Libraries like mlpack use this ideal of simplistic, modular design to ease onboarding for new users and to provide painless integration with existing software development processes.

We propose to lower the barrier for spaceflight deployments of machine learning further, by improving the accessibility and discoverability of mlpack and related projects in the C++ data science ecosystem. We will take a two-pronged approach:

(1) Robust support for a variety of spaceflight-grade hardware solutions. Although mlpack has support for cross-compilation to a wide variety of devices, for some computing platforms used in spaceflight applications, it does not provide a turn-key solution, and it may not take advantage of all possible hardware accelerations. We will rectify this by adding out-of-the-box configuration tooling for common spaceflight computing platforms, and supporting specific hardware accelerations on both CPUs and GPUs:

- *Improved CMake toolchain support for cross-compilation.* Currently mlpack provides direct support for a collection of boards including Raspberry Pis, Nvidia Jetsons, and RISC systems. Cross-compilation for these devices requires only specifying a single CMake flag. We will expand the set of directly supported devices to include a collection of COTS spaceflight computing solutions, including those mentioned earlier.

- *BF16/FP16 support for mlpack and Armadillo.* To support low-power machine learning, the use of low-precision numeric formats is common, either with 16-bit IEEE754 (FP16) or the more recent 'brain floating point' (BF16) formats [26, 33]. We will extend the Armadillo linear algebra library [55] to include FP16/BF16 support, both via recent low-precision extensions to OpenBLAS [61] and custom implemenations as needed. Due to mlpack's modularity [19], deployment of BF16 models will be trivial.

- *Expanded GPU support via Bandicoot.* mlpack already has support for some GPU machine learning algorithms via the CUDA and OpenCL backends of the Bandicoot GPU linear algebra library [20]. These implementations are tuned towards high-power desktop GPUs. We will extend the support of Bandicoot to low-power GPUs such as the ARM Mali line, by adding support for low-precision floating point types (FP16/BF16/FP8/BF8) and tuning Bandicoot's implementations for low-power GPUs.

(2) Demonstrations, examples, showcases, and documentation. When developing a solution, practitioners often try to start with examples (for instance, a Stack Overflow snippet [60]), and adapt them to the needs of their situation. As such, we will produce a significant number of turnkey mlpack solutions that will enable users to quickly get started.

- *Step-by-step deployment tutorials.* To assist prospective users, we will develop at least five end-to-end tutorials, structured as a narrative walkthrough of the process of deploying a prototype to a low-resource device. Each tutorial will target different hardware and use a different machine learning model class. All tutorials will be motivated by real-world spaceflight machine learning applications, including existing spaceflight uses of mlpack [27, 28]. These will be made available as online posts (e.g. on Medium), video tutorials (e.g. on Youtube), or standalone PDFs (e.g. on arXiv).

- *Showcases and examples.* Since some users prefer to start specifically with working code and adapt as necessary, we will enhance the mlpack examples repository [38] to include at least five additional fully-working examples demonstrating the use of mlpack on low-resource devices. All code will be fully commented to allow fast adaptation.

Task	Time estimate
Improved CMake toolchain support	80 hours
BF16/FP16 support for Armadillo	450 hours
Expanded GPU support via Bandicoot	450 hours
Step-by-step deployment tutorials	300 hours
Showcases and examples	300 hours
User-facing documentation improvement	300 hours
Seminar preparation/delivery	80 hours
Total	*2000 hours*

Table 1: Expected time estimates for each proposed work item.

- *User-facing documentation improvement.* When adapting examples to a new use case, users depend on high-quality API documentation to figure out how to change their code. We will review mlpack's existing API documentation, filling in any gaps for undocumented methods and adding notes as necessary for embedded applications.

- *Seminars for interested NASA practitioners.* We will coordinate with NASA Field Centers and Development Centers (e.g., JPL, GSFC, etc.) to identify groups who are using mlpack already or who could benefit from mlpack, and provide in-person training sessions or webinars to help them achieve their science goals. In addition to being a direct help to NASA groups, we also expect to gather important information about relevant development directions and needs that we can address in future work.

Timing and resource breakdown. We plan for two long-term mlpack contributors to perform the work proposed, both at a rate of 0.5 FTE for one year, totaling 2000 hours of work. Table 1 shows the expected time costs of each component of our proposal.

Project management. mlpack is a community-led open source project licensed under the permissive 3-clause BSD license [45], and accepts contributions from anyone. Significant decisions are done by simple majority vote from project contributors. Development is done on Github, and mlpack documentation and resources can be found on the mlpack website [37].

Impact and conclusion. The huge amount of data being generated by space science missions necessitates data analysis and machine learning directly on spacecraft using low-power, resource-constrained hardware. Expanding mlpack's support to cover this wide range of devices and creating ready-to-use turnkey examples and case studies directly enables scientists supporting the SMD to deploy advanced machine leaare using.rning solutions to spaceflight computing systems. Our proposal advances the accessibility and discoverability of mlpack and the wider low-resource C++ data science ecosystem, and directly develops open scientific analysis platforms, in line with the TOPS goals of the SMD's OSSI support. We envision—and are not far away from!—a landscape where scientists do not have to navigate an ever-changing, mystifying matrix of incompatible hardware and software, and can write their machine learning applications in a simple way and deploy them to whatever hardware they are using.

References

[1] Python package index - pypi.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for Large-Scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[3] Advanced Micro Devices, Inc. TensorFlow Installation, Accessed 2024.

[4] Apple Inc. Metal, Accessed 2024.

[5] ARM Community Blogs. Deploying Convolutional Neural Network on Cortex-M with CMSIS-NN, Accessed 2024.

[6] ARM Limited. ARM Cortex Microcontroller Software Interface Standard (CMSIS), Accessed 2024.

[7] Arm Limited. ArmNN, Accessed 2024.

[8] ARM Limited. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs, Accessed 2024.

[9] BAE Systems. Analytics and Machine Learning, Accessed 2024.

[10] BAE Systems. Radiation hardened electronics, Accessed 2024.

[11] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[12] Sriram Baireddy, Sundip R Desai, James L Mathieson, Richard H Foster, Moses W Chan, Mary L Comer, and Edward J Delp. Spacecraft time-series anomaly detection using transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1951–1960, 2021.

[13] Paul Bauman, Noel Chalmers, Nick Curtis, Chip Freitag, Joe Greathouse, Nicholas Malaya, Damon McDougall, Scott Moe, René van Oostrum, Noah Wolfe, et al. Introduction to amd gpu programming with hip. *Presentation at Oak Ridge National Laboratory. Online at: https://www. olcf. ornl. gov/calendar/intro-to-amd-gpu-programming-with-hip*, 2019.

[14] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[15] Nick Bild. Nothing but blue skies ahead for edge ML, Accessed 2024.

[16] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[17] Business Wire. CAES' Quad-Core LEON4FT Processor Selected for Next-Generation On-orbit Servicing Spacecraft, July 2022.

[18] Nolan Coulter and Hever Moncayo. An online machine learning paradigm for spacecraft fault detection. In *AIAA Scitech 2021 Forum*, page 1339, 2021.

[19] Ryan R Curtin and Marcus Edel. Designing and building the mlpack open-source machine learning library. *arXiv preprint arXiv:1708.05279*, 2017.

[20] Ryan R Curtin, Marcus Edel, and Conrad Sanderson. Bandicoot: C++ library for gpu linear algebra and scientific computing. *arXiv preprint arXiv:2308.03120*, 2023.

[21] Ryan R. Curtin, Marcus Edel, Omar Shrit, Shubham Agrawal, Suryoday Basak, James J. Balamuta, Ryan Birmingham, Kartik Dutt, Dirk Eddelbuettel, Rishabh Garg, Shikhar Jaiswal, Aakash Kaushik, Sangyeon Kim, Anjishnu Mukherjee, Nanubala Gnana Sai, Nippun Sharma, Yashwant Singh Parihar, Roshan Swain, and Conrad Sanderson. mlpack 4: a fast, header-only c++ machine learning library. *Journal of Open Source Software*, 8(82):5026, 2023.

[22] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. TensorFlow lite micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.

[23] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 8.4.0 manual: a high-level interactive language for numerical computations*, 2023.

[24] European Space Agency (ESA). LEON's first flights, January 2013.

[25] Gaisler Research. LEON3 Processor, Accessed 2024.

[26] Google Cloud. Bfloat16: The Secret to High Performance on Cloud TPUs, Accessed 2024.

[27] Timothy M Hackett, Sven G Bilén, Paulo Victor R Ferreira, Alexander M Wyglinski, and Richard C Reinhart. Implementation of a space communications cognitive engine. In *2017 Cognitive Communications for Aerospace Applications Workshop (CCAA)*, pages 1–7. IEEE, 2017.

[28] Timothy M Hackett, Sven G Bilén, Paulo Victor Rodrigues Ferreira, Alexander M Wyglinski, Richard C Reinhart, and Dale J Mortensen. Implementation and on-orbit testing results of a space communications cognitive engine. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):825–842, 2018.

[29] Honeywell Aerospace. Spacecraft on-board computer, Accessed 2024.

[30] The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022.

[31] Intel Corporation. Intel Extension for PyTorch, Accessed 2024.

[32] Intel Corporation. Intel® oneAPI, Accessed 2024.

[33] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[34] David Y Kusnierkiewicz. An overview of the timed spacecraft. *The Johns Hopkins University APL Technical Digest*, 24:150–155, 2003.

[35] David Y Kusnierkiewicz, Chris B Hersman, Yanping Guo, Sanae Kubota, and Joyce McDevitt. A description of the pluto-bound new horizons spacecraft. *Acta Astronautica*, 57(2-8):135–144, 2005.

[36] David C. McComas. Lessons from 30 years of flight software. In *MIT Lincoln Labs Software Engineering Symposium 2015*, number GSFC-E-DAA-TN26758, 2015.

[37] mlpack developers. mlpack: fast, header-only C++ machine learning library, Accessed 2024.

[38] mlpack developers. mlpack/examples: Example code for the mlpack library, Accessed 2024.

[39] Moog Inc. Moog BRE440 RAD-Hard CPU Datasheet, Accessed 2024. Available at: https://www.moog.com/content/dam/moog/literature/sdg/space/avionics/moog-BRE440-RADHardCPU-Datasheet.pdf.

[40] NASA Jet Propulsion Laboratory. Machine Learning and Instrument Autonomy Group, Accessed 2024.

[41] National Aeronautics and Space Administration. NASA Strategic Plan 2022. Technical Report NPD 1001.0D, National Aeronautics and Space Administration, 2022.

[42] National Aeronautics and Space Administration, Strategic Data Management Working Group. NASA Science Mission Directorate's Strategy for Data Management and Computing for Groundbreaking Science 2019–2024. Technical report, National Aeronautics and Space Administration, 2023.

[43] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008.

[44] OCE Technology. HiSAoR Rad-tolerant AI SoC, Accessed 2024.

[45] Open Source Initiative. BSD 3-Clause License, Accessed 2024.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Gregory Piatetsky. Python eats away at r: Top software for analytics, data science, machine learning in 2018: Trends and analysis. *KDnuggets. KDnuggets*, 2018.

[49] PyTorch Contributors. PyTorch Executorch Overview, Accessed 2024.

[50] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021.

[51] Gregg Rabideau and Edward Benowitz. Prototyping an onboard scheduler for the Mars 2020 rover. In *International Workshop on Planning and Scheduling for Space*, pages 1–9, 2017.

[52] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020.

[53] Eric S. Raymond. *The art of Unix programming*. Addison-Wesley Professional, 2003.

[54] Dennis M. Ritchie and Ken Thompson. The unix time-sharing system. *Bell System Technical Journal*, 57(6):1905–1929, 1978.

[55] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[56] Lindsay I. Smith. A tutorial on Principal Components Analysis. Technical Report OUCS-2002-12, Department of Computer Science, University of Otago, 2002.

[57] Jason Swope, Faiz Mirza, Emily Dunkel, Zaid Towfic, Steve Chien, Damon Russell, Joe Sauvageau, Doug Sheldon, Mark Fernandez, and Carrie Knox. Benchmarking remote sensing image processing and analysis on the snapdragon processor onboard the international space station. In *IGARSS 2022-2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 5305–5308. IEEE, 2022.

[58] Synova Inc. Synova Mongoose V, 2005. Archived webpage.

[59] TensorFlow Authors. Tensorflow lite, Accessed 2024.

[60] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, 24:637–673, 2019.

[61] Zhang Xianyi, Wang Qian, and Zhang Yunquan. Model-driven level 3 blas performance optimization on loongson 3a processor. In *2012 IEEE 18th international conference on parallel and distributed systems*, pages 684–691. IEEE, 2012.

[62] Maciej Ziaja, Piotr Bosowski, Michal Myller, Grzegorz Gajoch, Michal Gumiela, Jennifer Protich, Katherine Borda, Dhivya Jayaraman, Renata Dividino, and Jakub Nalepa. Benchmarking deep learning for on-board space applications. *Remote Sensing*, 13(19):3981, 2021.