

An accessible infrastructure for artificial intelligence using a docker-based Jupyterlab in Galaxy

Anup Kumar¹, Gianmauro Cuccuru¹, Björn Grüning¹, and Rolf Backofen^{1,2}

¹Bioinformatics Group, Department of Computer Science, University of Freiburg, Georges-Koehler-Allee 106, 79110 Freiburg, Germany

²Signalling Research Centres BIOS and CIBSS, University of Freiburg, Schaenzlestr. 18, 79104 Freiburg, Germany

Abstract: Artificial intelligence (AI) programs that train on a large amount of data require powerful compute infrastructure. Jupyterlab notebook provides an excellent framework for developing AI programs but it needs to be hosted on a powerful infrastructure to enable AI programs to train on large data. An open-source, docker-based, and GPU-enabled jupyterlab notebook infrastructure has been developed that runs on the public compute infrastructure of Galaxy Europe for rapid prototyping and developing end-to-end AI projects. Using such a notebook, long-running AI model training programs can be executed remotely. Trained models, represented in a standard open neural network exchange (ONNX) format, and other resulting datasets are created in Galaxy. Other features include GPU support for faster training, git integration for version control, the option of creating and executing pipelines of notebooks, and the availability of multiple dashboards for monitoring compute resources. These features make the jupyterlab notebook highly suitable for creating and managing AI projects. A recent scientific publication that predicts infected regions of COVID-19 CT scan images is reproduced using multiple features of this notebook. In addition, colabfold, a faster implementation of alphafold2, can also be accessed in this notebook to predict the 3D structure of protein sequences. Jupyterlab notebook is accessible in two ways - first as an interactive Galaxy tool and second by running the underlying docker container. In both ways, long-running training can be executed on Galaxy's compute infrastructure. The scripts to create the docker container are available under MIT license at <https://github.com/anuprulez/ml-jupyter-notebook>.

Contact:

kumara@informatik.uni-freiburg.de

anup.rulez@gmail.com

Keywords:

Jupyterlab; Jupyterlab Notebooks; Galaxy; Artificial intelligence; Remote model training;

Supplementary Note 1: Findings

A. Background. Bioinformatics comprises many sub-fields such as single-cell, medical imaging, sequencing, proteomics and many more that produce a huge amount of biological data in myriad formats. For example, the single-cell field creates gene expression patterns for each cell that are represented as matrices of real numbers. The medical imaging field generates images of cells and tissues, radiography im-

ages such as chest x-rays and computerized tomography (CT) scans. Next-Generation sequencing generates deoxyribonucleic acid (DNA) sequences that are stored as fasta (1) files. Artificial intelligence (AI) approaches such as machine learning (ML) and deep learning (DL) have been vastly used with these datasets (2) for predictive tasks such as medical diagnosis, imputing missing datasets, augmenting datasets, and estimating gene expression patterns and many more. To be able to use ML and DL algorithms on such datasets, a robust and efficient compute infrastructure is needed that can serve multiple purposes. They include pre-processing raw datasets to transform them into suitable formats that are understood by ML and DL algorithms, creating and executing complex architectures of ML and DL algorithms on pre-processed datasets and making models and predicted datasets readily available for further analyses. To facilitate such tasks, a complete infrastructure is developed that combines jupyterlab (3) notebook, augmented with many useful features, running on public compute resources of Galaxy (4) Europe to perform end-to-end AI analysis on biological datasets. The infrastructure consists of three important components. First, a docker container that encapsulates jupyterlab together with packages such as git (5), elyra AI (6), tensorflow GPU (7), CUDA (8) for tensorflow to interact with GPU and many others. Second, a Galaxy interactive tool (9, 10) downloads this docker container to serve jupyterlab online on Galaxy Europe. Third, the compute infrastructure, consisting of several CPUs and GPUs, on which the online jupyterlab runs.

B. Jupyterlab. Jupyterlab is a web-based, robust editor used for varied purposes such as data science, scientific computing, machine learning and deep learning. It's a common editor that supports more than 40 programming languages, some of the popular ones are python, R, julia and scala. Python is one of the most popular languages used by researchers for performing numerous scientific and predictive analyses. Therefore, it is used as the programming language in Galaxy jupyterlab interactive tool because many popular packages such as tensorflow, pytorch (11), scikit-learn (12) for machine learning and deep learning are readily available as python packages. Moreover, the extensible architecture of jupyterlab makes it possible to add many external packages and plugins such as git, dashboards and many others. Editors such as jupyterlab, integrated with several useful packages, provide

a favourable platform for both, rapid prototyping and end-to-end development and management of AI projects. To harness the benefits of jupyterlab, it has been used as the editor for an interactive tool in Galaxy.

C. Docker container. Docker (13) containers are popular for shipping packaged software as complete ecosystems enabling them to be reproducible in a platform-independent manner. Software executing inside a docker container is abstracted from the operating system (OS) as most of the requirements necessary for them to run successfully are already configured inside the container. A container runs as an isolated environment making the minimum number of interactions to the host OS and thereby, improving the security aspects of running software. Docker container is used in this project to encapsulate jupyterlab along with many useful packages such as CUDA, tensorflow, ONNX (14), scikit-learn and many others. Docker container inherits many packages such as numpy (15), scipy (16) from its base container, Jupyter/tensorflow-notebook (17), and augments it with many other packages suitable for machine learning, deep learning and visualisation. Docker container is independent of Galaxy and can be separately executed for serving jupyterlab with the same set of packages on a different compute infrastructure or any personal computer (PC) and laptop. Moreover, it can easily be extended by installing suitable packages only by adding their appropriate package names in its dockerfile (18). The docker container then should be rebuilt and uploaded to the docker hub (19). Galaxy interactive tool can automatically download the updated container from the docker hub and the newly installed packages become available in the jupyterlab notebook.

D. Features. Many features such as easy accessibility, support of a wide variety of programming languages of jupyterlab, and extensibility to install useful plugins make it a desirable editor for researchers, especially for creating prototypes rapidly. In this project, many such features have been integrated into a jupyterlab notebook which is served online on Galaxy Europe with large compute resources to enable researchers to create prototypes and end-to-end AI projects (Figure 1). Some of the important features of jupyterlab hosted on Galaxy Europe are GPU support for faster execution of deep learning programs. Tensorflow-GPU interacts with GPU using CUDA when the backend compute resource has GPU(s), otherwise, the program in a jupyterlab notebook runs on CPUs. Other useful packages include ONNX for transforming trained tensorflow and scikit-learn models to ONNX models, open-CV (20) and scikit-image (21) for processing images, nibabel (22) for reading image files stored as “.nii”, bioblend (23) for connecting to Galaxy to access its datasets, histories and workflows in a jupyterlab notebook and visualisation package such as bqplot (24) for plotting interactive charts, voila (25) for displaying output cells of a jupyterlab notebook, dashboard such as nvdashboard (26) for monitoring GPU performance. Support for the file extension such as H5 (27), efficient for storing matrices, enables machine learning researchers to save model weights and input

datasets for an AI algorithm. Other packages such as colabfold (28) together with JAX (29) are used for predicting 3D structures of proteins which are discussed in detail in a later section. In addition, it is possible to create a long-running training job that runs on a remote compute infrastructure and the trained models and output datasets get stored permanently in Galaxy history. The trained model is saved as an ONNX file and tabular datasets in H5 file.

E. Related infrastructure. There are a few infrastructures available, free and commercial, that offer jupyterlab or similar environments for developing data science and AI projects. A few popular ones are google colab (30), kaggle kernels (31) and amazon’s sagemaker (32). Google colab is partially free and offers an online editor similar to a jupyterlab notebook. The free version of colab offers dynamic compute resources. The disk space is around 70 gigabytes (GB) and memory (RAM) is around 12 GB. These resources are scarce for projects that deal with high-dimensional biological data (33, 34). These resources are also variable and depend on past usage. More compute resources are assigned to those users that have used less in the past for a more equitable sharing of resources. Moreover, there is a limit of the running time of only 12 hours which is also insufficient and not ideal for long-running model learning training on large datasets. However, colab pro and pro+ offer better compute resources but they come at a price - EUR 9.25 and EUR 42.25 per month, respectively. In contrast, kaggle kernels are free of charge but similar to colab, their computing resources are scarce. The total disk space is approximately 73 GB and RAM is 16 GB for a CPU-based kernel. For the GPU-based kernel, the disk space is of the same size as that of the CPU-based kernel but the RAM of the CPU decreases to 13 GB and an additional RAM of 15 GB is added through GPU and computation time is limited to 30 hours a week. It also supports TPUs but the computation time is even more limited to only 20 hours a week. Amazon’s sagemaker is also a commercial software for developing AI algorithms that is free of charge but only for 2 months. Overall, these notebook infrastructures do not offer unrestricted compute resources free of charge. To address the drawbacks of these notebook servers and provide researchers and users large compute resources more reliably, Galaxy jupyterlab offers 1 terabyte (TB) of disk space and unlimited computation time on 1 GPU and 7 CPUs per session. RAM for GPU is around 15 GB and for CPUs is 20 GB (Table 1). The offered resources in the jupyterlab notebook running in Galaxy stay constant and are independent of past usage. To make it more useful, jupyterlab opens a tab for each notebook that allows researchers to develop and execute all notebooks inside the same session of the allotted computational resource rather than forcing them to connect to a different session as in google’s colab and kaggle kernels.

Supplementary Note 2: Implementation

Jupyterlab infrastructure has been developed in two stages. First, a docker container is created containing all the neces-

sary packages such as jupyterlab itself, CUDA for interaction with GPU, tensorflow, scikit-learn, ONNX and so on. The docker container is inherited from a base container that is suited for serving entire jupyterlab environments. In addition to software packages such as numpy, scipy, and tensorflow that are already wrapped around the base container, many packages are added with their compatible versions. Compatible packages for CUDA, CUDA DNN and tensorflow are necessary so that they together interact with the GPU on the host machine for accelerating deep learning programs. Other significant packages, integrated into the docker container, are ONNX, scikit-learn, elyra AI, bioblend, nibabel, scikit-image, open-CV, bqplot and voila. The integrated docker container contains all the necessary packages for developing data science, machine learning and deep learning projects. Second, the container can be downloaded to any powerful compute infrastructure and jupyterlab can be served in a browser via the URL it creates. In addition, to run this container in Galaxy, an interactive tool is created that downloads this container on a remote compute infrastructure and generates a URL that is used to run jupyterlab in a browser. The architecture of jupyterlab infrastructure in Galaxy is shown in Figure 1.

The running instance of Jupyterlab in Galaxy contains a home page, a jupyterlab notebook, that summarises several of its features. Further, there are other notebooks available, each describing a feature of the jupyterlab with code examples such as how to create ONNX models for scikit-learn and tensorflow classifiers, how to connect to Galaxy using bioblend, how to create interactive plots using bqplots and how to create a pipeline of notebooks using elyra AI. To access the jupyterlab notebook in Galaxy Europe, a ready-to-use hands-on GTN (35) tutorial (36) has been developed that shows steps such as opening the notebook, using git to clone codebase from github, and sending long-running training jobs to a remote Galaxy cluster. The two use-cases, explained in previous sections, are also covered in the tutorial along with their respective code scripts as separate jupyterlab notebooks.

.1. Remote model training. For large datasets, model training may need several hours or even days. In such cases, it would be non-ideal to keep the jupyterlab notebook open in a browser's tab till the training finishes. Therefore, another Galaxy tool (37) is developed to enable researchers to send long-running training jobs to a remote Galaxy cluster. The tool can be executed from a jupyterlab notebook using a custom python function (38) that takes input datasets and training script as input parameters. The input datasets to be used for training, testing and validation must be provide in H5 format. This is done to standardise input data format for AI models that train on matrices as input data can be in multiple formats such as images, genomic sequences, gene expression patterns. H5 files can be created using any of these data formats and fed to the AI model. The long-running training happens in a remote cluster in Galaxy as a regular job. Upon completion of the job, the resulting datasets and the trained model become available in a newly created Galaxy

history. This feature of outsourcing deep learning's long-running training to a remote cluster decouples it from the jupyterlab notebook. The trained model and other datasets can be downloaded from the Galaxy history for further analysis.

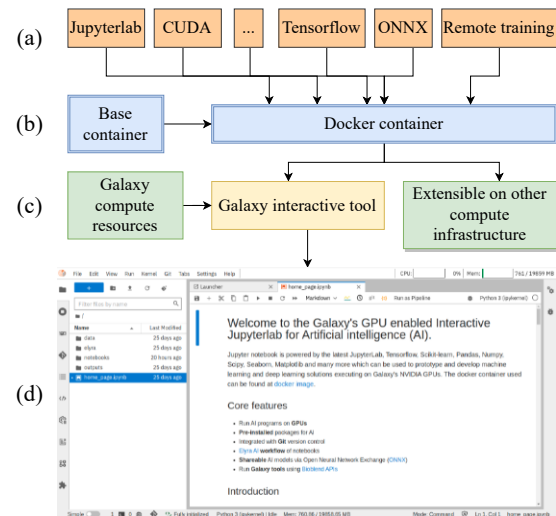


Fig. 1. Architecture of Galaxy's jupyterlab. Part (a) shows packages and features wrapped inside a docker container. Part (b) shows a base docker container (17) from which the customized container (19) is derived. In part (c) Galaxy's interactive tool downloads the customized container. The customized docker container can also be hosted on a different compute infrastructure. Part (d) shows Galaxy's jupyterlab.

Supplementary Note 3: Results

Jupyterlab infrastructure in Galaxy is used to reproduce two scientific publications that demonstrate its power to develop deep learning models using COVID CT scan images (39) and predict the 3D structure of proteins using colabfold, a faster implementation of alphafold2 (40).

A. COVID-19 CT scan image segmentation. In (39), COVID-19 CT scan images have been used to develop and train a deep learning model architecture that predicts COVID-19 infected regions in CT scan images with high accuracy. An open-source implementation of the work is available that trains a unet deep learning architecture that distinguishes between normal and infected regions in CT scans. The code of this implementation is adapted and executed on our jupyterlab notebook infrastructure. The CT scan images used in the original work (39) are transformed into an H5 file so that they can directly be used as an input to the unet architecture defined in the jupyterlab notebook (41). A composite H5 file (42) is created using script (43) that contains multiple datasets inside and each dataset is a matrix corresponding to the matrices training, test and validation as used in the original work. The entire analysis of the original work can be reproduced using multiple notebooks in (41) by achieving similar values of precision and recall (approximately 0.98) as mentioned in the original work. In (41), the first notebook (1_fetch_datasets.ipynb) downloads input dataset as an H5 file. Additionally, it also

Table 1. Comparison with other notebook infrastructures

Indicators/Infrastructures	Google Colab	Kaggle Kernel	Galaxy Jupyterlab
Memory/Disk space (GB)	12/70	16/73	20/1000
GPU/TPU	Yes/ Yes	Yes/ Yes	Yes/No
Max usage time (Hours)	12	12, 30 hrs of GPU/week, 20 hrs of TPU/week	No time restriction on GPU usage, note
Dynamic compute resources	Yes	Yes	Fixed and guaranteed
Remote model training	No	No	Yes

downloads the trained ONNX model. The second notebook (`2_create_model_and_train.ipynb`) creates and trains a unet model on the training dataset extracted from the H5 file. Training, accelerated by GPU, for 10 iterations over the entire training dataset finishes quickly. The third notebook (`3_predict_masks.ipynb`) extracts the test dataset and predicts infected regions of the CT scans in the test dataset using the trained model created by the second notebook. Figure 2 shows the comparison of ground truth infected regions (second column) and the predicted infected regions in the third and fourth columns. Original CT scans from the test dataset are shown in the first column of Figure 2.

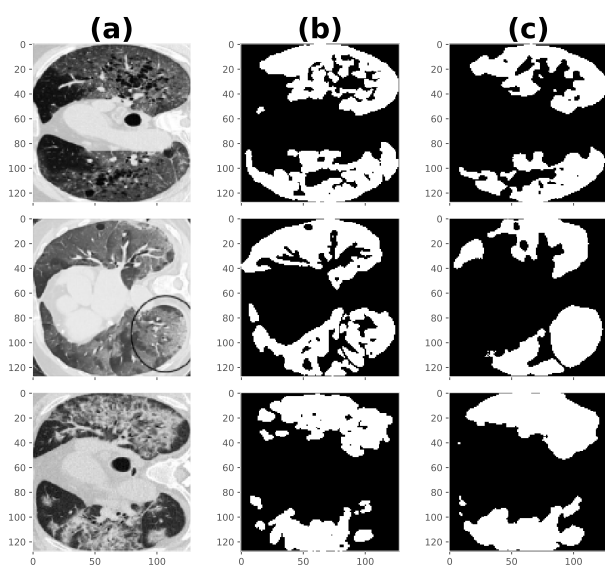


Fig. 2. Figures shows original CT scans in column (a), corresponding ground-truth masks of original CT scans in column (b) and the predicted masks in column (c). Masks are COVID-19 infected regions in the corresponding CT scans. The ground-truth and predicted masks show high similarity.

In (41), the notebook "`4_create_model_and_train_remote.ipynb`" defines the entire script for developing and training the unet architecture. "`5_run_remote_training.ipynb`" notebook executes the previous notebook on a cluster remotely after creating a Galaxy history and then uploading the script extracted from "`4_create_model_and_train_remote.ipynb`" notebook and datasets. The custom python function (`run_script_job`) creates a Galaxy history using bioblend and then uploads the datasets to the same history. After the upload is finished, the python script from the specified notebook is executed

dynamically. It trains a deep learning model on the uploaded datasets to create a model which is saved as an ONNX model in Galaxy history. Using the notebook "`6_predict_masks_remote_model.ipynb`" from (41), the trained model can be downloaded from the Galaxy history and used for predicting infected regions of the CT scans of the test dataset. A significant advantage of training deep learning models remotely is that researchers don't have to keep the jupyterlab notebook session running as long as the model is being trained. It provides a convenient way for training models, especially those that take several hours or even days to finish.

A.1. Predict 3D structure of proteins using ColabFold. Alphafold2 has made a breakthrough in predicting the 3D structure of proteins with outstanding accuracy. However, due to their large database size (a few TB), it is not easily accessible to researchers. Therefore, a few approaches have been developed that replace the time-consuming steps of alphafold2 with slightly different steps but predict the 3D structure of proteins with similar accuracy while consuming less memory and time. One such approach is colabfold which replaces a large database search in alphafold2 for finding homologous sequences by a significantly (40-60 times) faster MMseqs2 API (44) call to generate input features based on the query protein sequence. Colabfold's prediction of 3D structures in batches is approximately 90 times faster. Colabfold is integrated into the docker container (19) by adding two packages - colabfold and GPU-enabled JAX which is a just-in-time compiler for making mathematical transformations. "`7_ColabFold_MMseq2.ipynb`" notebook in (41) makes the prediction of 3D structure using colabfold by making use of the alphafold2 pre-trained weights. Figure 3 shows the 3D structure of 4Oxalocrotonate_Tautomerase (45), a protein sequence of length 62, along with its side chains. This 3D structure is extremely similar to the structure predicted by the jupyter notebook (46) from colabfold in (28).

Supplementary Note 4: Summary

Jupyterlab notebook is integrated as an interactive tool in Galaxy Europe running on a powerful compute infrastructure comprising CPUs and GPUs. Jupyterlab is configured in a docker container along with many different packages such as CUDA, tensorflow, scikit-learn, and elyra AI to provide a robust architecture for the development and management of

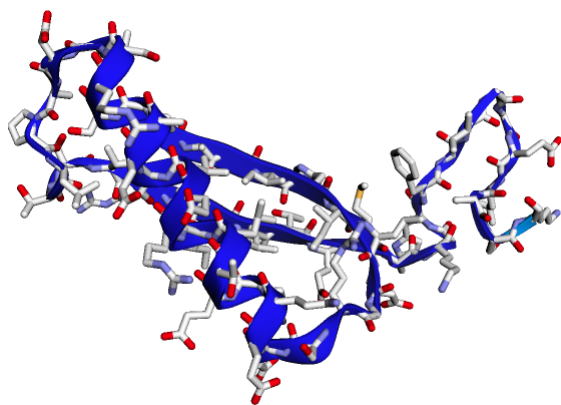


Fig. 3. Figures shows a 3D structure of 4 Oxalocrotonate_Tautomerase enzyme (protein) predicted by ColabFold

projects from data science, machine learning and deep learning. Remote training makes it convenient to run multiple analyses in parallel as different Galaxy jobs by executing the same Galaxy tool and results become available in different Galaxy histories. Features such as git integration are useful for managing entire code repositories on github and elyra AI for creating pipelines of notebooks to be executed as one unit of software. All notebooks run on the same session of the jupyterlab. The entire infrastructure of jupyterlab is readily accessible through Galaxy Europe. In contrast to commercial infrastructures that host editors similar to Jupyterlab and offer powerful and reliable compute only through paid subscriptions, this infrastructure provides large compute resources invariant to usage and has unlimited usage time while ensuring the same set compute resources for multiple usages.

Supplementary Note 5: Availability of supporting source code and requirements

Project name: GPU-enabled docker container with Jupyterlab for artificial intelligence

Project home page: <https://github.com/anupruez/ml-jupyter-notebook>

Galaxy interactive tool: https://github.com/usegalaxy-eu/galaxy/blob/release_22.01_europe/tools/interactive/interactivetool_ml_jupyter_notebook.xml

Operating system: Linux

Programming languages: Python, XML, Docker, Bash

Other requirements:

License: MIT License

Biotools ID: `gpu-enabled_docker_container_with_jupyterlab_for_ai`

Supplementary Note 6: Declarations

A. List of abbreviations. GB: Gigabyte; AI: Artificial intelligence

B. Competing interests. The authors declare that they have no competing interests.

Author Approvals

All authors have seen and approved the manuscript, and it hasn't been accepted or published elsewhere.

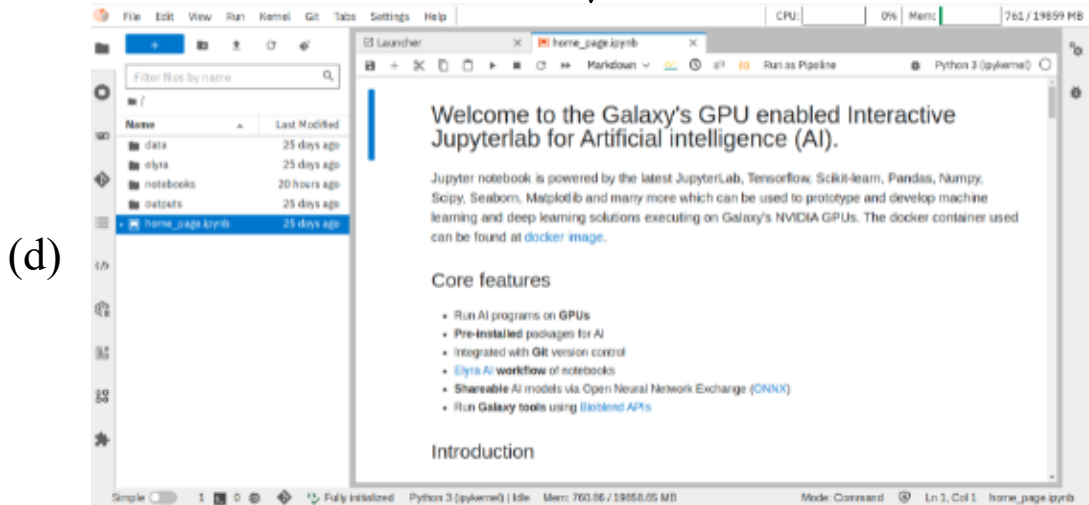
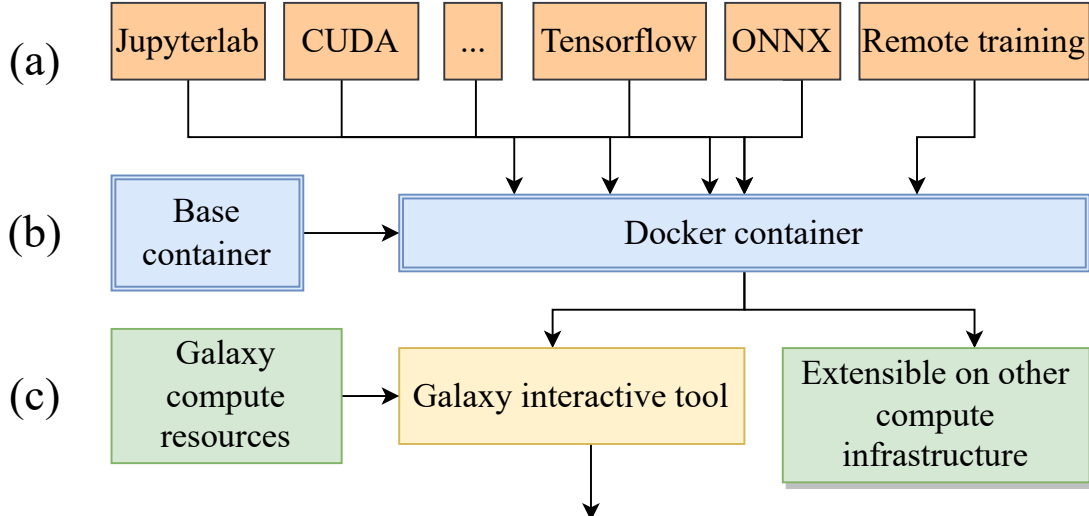
C. Authors' contributions. Authors' contributions follow the order of names.

D. Acknowledgements. We thank Galaxy Europe for its support.

Supplementary Note 7: References

1. W. Pearson and et. al. The fasta package - protein and dna sequence similarity searching and alignment programs. <https://github.com/wrpearson/fastas36>. 2016. Accessed 30 June 2022, 2016.
2. I. Kumar, S.P. Singh, and Shivam. Chapter 26 - machine learning in bioinformatics. *Academic Press*, pages 443–456, 2022. doi: <https://doi.org/10.1016/B978-0-323-89775-4.00020-1>.
3. Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. page 87, 2016.
4. E. Afgan, D. Baker, B. Batut, and et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1): W537–W544, 2018. doi: <https://doi.org/10.1093/nar/gky379>.
5. Frédéric et. al. (2017) Collonval. A jupyterlab extension for version control using git. <https://github.com/jupyterlab/jupyterlab-git>. 2017. Accessed 29 June 2022.
6. Luciano et. al. (2018) Resende. Elyra is a set of ai-centric extensions to jupyterlab notebooks. <https://github.com/elyra-ai/elyra>. 2018. Accessed 29 June 2022.
7. Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
8. NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89. <https://developer.nvidia.com/cuda-toolkit>. 2020. Accessed 29 June 2022, 2020.
9. Galaxy Europe. Live instance of the european galaxy server. <https://live.usegalaxy.eu/>. 2019. Accessed 30 June 2022, 2019.
10. Anup Kumar and et. al. Gpu enabled interactive jupyter notebook for machine learning. https://github.com/usegalaxy-eu/galaxy/blob/release_22.01_europe/tools/interactive/interactivetool_ml_jupyter_notebook.xml, 2021.
11. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.
12. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
13. Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.
14. Junjie Bai, Fang Lu, Ke Zhang, and et. al. Onnx: Open neural network exchange. <https://github.com/onnx/onnx>. 2019. Accessed 29 June 2022, 2019.
15. Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2).
16. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, António H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
17. Jupyter Project. Jupyter notebook scientific python stack w/ tensorflow. <https://hub.docker.com/r/jupyter/tensorflow-notebook>. 2014. Accessed 29 June 2022, 2014.
18. A. Kumar. Jupyter container used for data science and tensorflow. <https://github.com/anupruez/ml-jupyter-notebook/blob/master/Dockerfile>. 2021. Accessed 29 June 2022, 2021.

19. A. Kumar. Container for machine learning and deep learning in jupyter notebook. <https://hub.docker.com/repository/docker/anupkumar/docker-ml-jupyterlab>. 2021. Accessed 29 June 2022, 2021.
20. G. Bradski. The OpenCV Library. *Dr. Dobbs's Journal of Software Tools*, 2000.
21. Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.
22. Matthew Brett, Christopher J. Markiewicz, Michael Hanke, Marc-Alexandre Côté, Ben Cipollini, Paul McCarthy, Dorota Jarecka, Christopher P. Cheng, Yaroslav O. Halchenko, Michiel Cottaar, Eric Larson, Satrajit Ghosh, Damian Wassermann, Stephan Gerhard, Gregory R. Lee, Hao-Ting Wang, Erik Kastman, Jakub Kaczmarzyk, Roberto Guidotti, Or Duek, Jonathan Daniel, Ariel Rokem, Cindee Madison, Brendan Moloney, Félix C. Morency, Mathias Goncalves, Ross Markello, Cameron Riddell, Christopher Burns, Jarrod Millman, Alexandre Gramfort, Anibal Sólón, Jaakko Leppäkangas, Jasper J.F. van den Bosch, Robert D. Vincent, Henry Braun, Krish Subramaniam, Krzysztof J. Gorgolewski, Pradeep Reddy Raamana, Julian Klug, B. Nolan Nichols, Eric M. Baker, Soichi Hayashi, Basile Pinsard, Christian Haselgrove, Mark Hymers, Oscar Esteban, Serge Koudoro, Fernando Pérez-García, Nikolaas N. Oosterhof, Bago Amirbekian, Ian Nimmo-Smith, Ly Nguyen, Samir Reddigari, Samuel St-Jean, Dimitri Papadopoulos Orfanos, Egor Panfilov, Eleftherios Garyfallidis, Gael Varoquaux, Jon Haitz Legarreta, Kevin S. Hahn, Oliver P. Hinds, Bennet Fauber, Jean-Baptiste Poline, Jon Stutters, Kesshi Jordan, Matthew Cieslak, Miguel Estevan Moreno, Valentin Haenel, Yannick Schwartz, Zvi Baratz, Andrew Van, Benjamin C Darwin, Bertrand Thirion, Carl Gauthier, Igor Solovey, Ivan Gonzalez, Jath Palasubramaniam, Justin Lecher, Katrin Leinweber, Konstantinos Raktivan, Lea Waller, Markéta Calábková, Peter Fischer, Philippe Gervais, Syam Gadde, Thomas Ballinger, Thomas Roos, Tomáš Hrnčiar, Venkateswara Reddy Reddam, and freec84. nipy/nibabel: 3.2.2, June 2022.
23. Clare Sloggett, Nuwan Goonasekera, and Enis Afgan. BioBlend: automating pipeline analyses within Galaxy and CloudMan. *Bioinformatics*, 29(13):1685–1686, 2013. doi: 10.1093/bioinformatics/btt199.
24. Sylvain Corlay and et. al. 2-d plotting library for project jupyter. <https://github.com/bqplot/bqplot>. 2015. Accessed 29 June 2022, 2015.
25. Jeremy Tuloup and et. al. Rendering of live jupyter notebooks with interactive widgets. <https://github.com/voila-dashboards/voila>. 2018. Accessed 29 June 2022, 2018.
26. Jacob Tomlinson and et. al. A jupyterlab extension for displaying gpu usage dashboards. <https://github.com/rapidsai/jupyterlab-nvdashboard>. 2021. Accessed 29 June 2022, 2021.
27. The HDF Group. Hierarchical Data Format, version 5. <https://www.hdfgroup.org/HDF5/>. 1997. Accessed 29 June 2022, 1997-2022.
28. M. Mirdita, K. Schütze, Y. Moriwaki, and et al. ColabFold: making protein folding accessible to all. *Nat Methods*, 19:679–682 (2022), 2022. doi: 10.1038/s41592-022-01488-1.
29. Matthew Johnson and et. al. Jax: Autograd and xla. <https://github.com/google/jax>. 2020. Accessed 29 June 2022, 2020.
30. Ekaba Bisong. Google colaboratory. pages 59–64, 2019. doi: 10.1007/978-1-4842-4470-8_7.
31. Kaggle. Kaggle. <https://www.kaggle.com>. 2010. Accessed 29 June 2022, 2020.
32. Amazon SageMaker. Amazon sagemaker. <https://aws.amazon.com/sagemaker/>. 2017. Accessed 29 June 2022, 2017.
33. K.R. Moon, D. van Dijk, and Z. et al. Wang. Visualizing structure and transitions in high-dimensional biological data. *Nat Biotechnol*, 37:1482–1492 (2019), 2019. doi: 10.1038/s41587-019-0336-3.
34. Philippe Boileau, Nima S Hejazi, and Sandrine Dudoit. Exploring high-dimensional biological data with sparse contrastive principal component analysis. *Bioinformatics*, 36(11): 3422–3430, 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa176.
35. Bérénice Batut, Saskia Hilttemann, Andrea Bagnacani, Dannon Baker, Vivek Bhardwaj, Clemens Blank, Anthony Bretaudeau, Loraine Brillet-Guéguen, Martin Čech, John Chilton, Dave Clements, Olivia Doppelt-Azeroual, Anika Erxleben, Mallory Ann Freeberg, Simon Gladman, Youri Hoogstrate, Hans-Rudolf Hotz, Torsten Houwaart, Pratik Jagtap, Delphine Larivière, Gildas Le Corguillé, Thomas Manke, Fabien Mareuil, Fidel Ramírez, Devon Ryan, Florian Christoph Sigloch, Nicola Soranzo, Joachim Wolff, Pavankumar Videm, Markus Wolfen, Aisanjiang Wubuli, Dilmurat Yusuf, James Taylor, Rolf Backofen, Anton Nekrutenko, and Björn Grüning. Community-driven data analysis training for biology. *Cell Systems*, 6(6): 752–758.e1, jun 2018. doi: 10.1016/j.cels.2018.05.012.
36. Anup Kumar. A docker-based interactive jupyterlab powered by gpu for artificial intelligence in galaxy (galaxy training materials). https://training.galaxyproject.org/training-material/topics/statistics/tutorials/gpu_jupyter_lab/tutorial.html. 2022. Accessed 29 June 2022, 2022.
37. A. Kumar. Run long running jupyterlab script. https://github.com/bgruening/galaxytools/blob/master/tools/jupyter_job/run_jupyter_job.xml. 2022. Accessed 30 June 2022, 2022.
38. A. Kumar. Custom jupyterlab notebook function to start model training job in galaxy. https://github.com/anupruelez/ml-jupyter-notebook/blob/galaxy-inte-docker-ml-jupyter/galaxy_script_job.py#L43. 2021. Accessed 30 June 2022, 2021.
39. N. Saeedizadeh, S. Minaee, R. Kafieh, S. Yazdani, and M. Sonka. Covid tv-unet: Segmenting covid-19 chest ct images using connectivity imposed unet. *Computer Methods and Programs in Biomedicine Update*, 1:100007, 2021. ISSN 2666-9900. doi: 10.1016/j.cmpbup.2021.100007.
40. J. Jumper, R. Evans, A. Pritzel, and et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589 (2021), 2021. doi: 10.1038/s41586-021-03819-2.
41. A. Kumar. Jupyterlab notebooks. https://github.com/anupruelez/gpu_jupyterlab_ct_image_segmentation. 2022. Accessed 29 June 2022, 2022.
42. A. Kumar. COVID Image segmentation datasets and trained model, 2022.
43. N. Saeedizadeh, S. Minaee, R. Kafieh, S. Yazdani, and M. Sonka. Covid tv-unet: Segmenting covid-19 chest ct images using connectivity imposed unet. https://github.com/narges-sa/COVID-CT-Segmentation/blob/main/main_TV_Unet_Split1.py. 2021. Accessed 30 June 2022, 2021.
44. M. Steinegger and J. Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat Biotechnol*, 35:1026–1028 (2017), 2017. doi: 10.1038/nbt.3988.
45. Chen LH, Kenyon GL, Curtin F, Harayama S, Bembenek ME, Hajipour G, and Whitman CP. 4-oxalocrotonate tautomerase, an enzyme composed of 62 amino acid residues per monomer. *J Biol Chem*, 267(25):17716–21, 1992.
46. M. Mirdita, K. Schütze, Y. Moriwaki, and et al. ColabFold: making protein folding accessible to all. <https://github.com/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb>. 2022. Accessed 30 June 2022, 2022.



(a)**(b)****(c)**