Yerbamaté: An Open Science Python Framework

v0.9

Ali Rahimi

Maastricht University
Department of Advanced Computing Sciences
Maastricht, The Netherlands

***Abstract*—This report presentes Yerbamate[1], an open science framework designed for Python-based research projects, with a focus on Artificial Intelligence (AI) and deep learning. The framework prioritizes modularity and separation of concerns to enhance the developer experience, promote reproducibility, reusability, shareability, maintainability, and code quality. Yerbamate supports all Python frameworks and libraries, including NumPy, Jax, Flax, Tensorflow, Keras, PyTorch, and Hugging Face, and can be applied to a wide range of tasks, such as natural language processing, computer vision, deep learning, machine learning and optimizations. Yerbamate simplifies the experimentation process, providing a user-friendly installation procedure, intuitive commands for running experiments, and streamlined execution of code. The framework fosters open science by enabling reuse and sharing of independent code modules, such as models and trainers, among practitioners. Adoption of Yerbamate can facilitate collaboration, enhance the developer experience, and promote open science while contributing to addressing the reproducibility crisis in the field of AI.**

I. Introduction

In recent years, AI has made remarkable progress, finding numerous applications across various fields such as natural language ([1]), general intelligence([2]), art ([3]), music ([4]), healthcare ([5]), finance ([6]), education ([7]), and weather forecasting ([8]). The potential benefits of AI are immense ([9, 10]), their development and deployment require quality data ([11]), code, and sound software engineering practices ([12, 13]). Poor software engineering practices can lead to a reproducibility crisis, undermining the reliability and credibility of AI systems ([14]). On the other hand, good software engineering practices can enhance the developer experience, reliability, maintainability, and replicability of AI systems ([12, 13, 15]). In order to realize the full potential of AI, researchers and practitioners require open source tools and frameworks that are user-friendly, customizable, and promote reproducibility in the development of AI projects ([16, 17, 18, 19, 20, 21]). This is particularly important for ensuring that the benefits of AI are accessible to all and that the development of AI aligns with open science and ethical principles, enabling the sharing of research findings and supporting the progress and implementation of diverse AI applications for societal benefits ([22, 23, 24, 25, 20]).

The Yerbamate framework embodies a novel approach to open science and python development, showcasing a python modular project structure and software engineering design that enhances developer experience, reproducibility, reusability, transparency, and collaboration in the research community. This work incorporates contemporary open science, combining open source and open science methodologies by leveraging version control (git) for sharing reusable code modules and community-based discussion platforms for open access to open science work of Yerbamate including open collaboration, question answering, feedback, and issue tracking.

[1]https://github.com/oalee/yerbamate

II. Motivation

The reproducibility crisis in AI has led to an increasing focus on open science practices to improve the validity and reproduciblity of research ([22, 23, 26, 27, 28, 29]. Moreover, the gap of software engineering knowledge among researchers and practitioners often leads to poorly structured and documented AI projects, resulting in reproducibility issues, bugs, and invalidation of research. ([14, 30, 31, 32, 33]).

The absence of widely adopted software engineering principles in AI research has resulted in barriers to collaboration and building upon previous research ([34]). Open-source AI research projects often create their own command-line tools or frameworks to experiment with different models and hyperparameters, which can result in comprehensive options but also limitations. This often leads to the reinvention of the wheel, as each project implements the same tasks slightly differently for their specific needs.

This heterogeneity in software engineering practices poses a significant obstacle to code reuse and collaboration among researchers. In some cases, AI codes are of low quality, resembling spaghetti code that is hard to maintain ([31, 32, 13, 33, 14, 35]). In contrast, others exhibit a more modular design, which can enhance code quality and maintainability ([31, 32, 15]). Moreover, the problem arises when attempting to use another researcher's model, data augmentation, or a specific approach, as it requires learning a new framework for hyperparameter configuration and execution. The lack of standardized software engineering practices increases the difficulty of reproducing and building upon previous research.

III. Background Information

A. Crisis of Reproducibility in AI

The crisis of reproducibility in AI refers to the difficulty in reproducing the results of AI research ([21]). The lack

of transparency in data collection and research has greatly contributed to the crisis of reproducibility in AI ([21, 36, 14]). Many AI models are developed close sourced using proprietary data and methods, making it difficult for others to replicate the research and understand the inner workings of the models ([21, 34]). Additionally, the lack of transparency in the research process can lead to issues such as unreliable or biased methods data, which can further undermine the credibility and reproducibility of the research, and it can decrease the trust in the field as the results of the research are not independently verifiable ([34, 14, 37]). The pressure to publish results and the lack of incentives to share data and code can discourage researchers from making their work easily reproducible. ([38, 39, 40, 41, 42])

B. Open Science

Open science is a research methodology that prioritizes transparency, collaboration, and reproducibility ([43]). The promotion of open science in the field of AI has garnered considerable attention in recent years ([34, 21, 14, 37, 29, 22, 23, 26, 44]). In the field of AI, open science practices can help to address concerns about biased or unreliable data, as well as provide a way for researchers to collaborate and build reproducible research and enhance accountability in AI ([34, 29]). Open science encourages researchers to share their knowledge, data, code, and detailed documentation of their methods ([36, 34]). Researchers can also use open-source frameworks and standard evaluation metrics ([21]) to facilitate reproducibility. Furthermore, the scientific community can encourage reproducibility by valuing it in the peer-review process ([37]), and by giving credit to researchers who share their data and code ([37, 45, 29]).

C. The Significance of Open Science for AI

Open science represents a crucial component in the pursuit of responsible and trustworthy AI ([46, 22, 23, 26]). By prioritizing transparency and reproducibility, researchers in the field can advance its development in a safer and trustworthy manner ([22, 25, 28, 29]). Open science practices serve to mitigate the risks associated with closed-source AI and big data bias, which have raised significant concerns among stakeholders ([47, 48]). Adoption of open science and open source AI can increase the fairness and impartiality of AI models ([29, 34, 21]) and enhance the credibility and trustworthiness of their outputs among stakeholders and decision-makers ([49, 50, 51]).

D. Fair Comparison of Models in AI

Data is a crucial factor in the success of deep learning models ([11]). The quality, pre-processing, and augmentations applied to the data can significantly impact the model's ability to extract knowledge and make accurate predictions ([52]). Therefore, it is essential for researchers to consistently use the same data split, pre-processing and augmentations when comparing models to ensure fair comparisons ([53, 54, 14]).

E. The FAIR Guiding Principles for scientific data management and stewardship

The FAIR Guiding Principles for scientific data management and stewardship were published in Scientific Data, an online Nature magazine journal in 2016 as a guideline for improving the findability, accessibility, interoperability, and reusability of digital assets, with a focus on machine-actionability to enable computational systems to find, access, interoperate, and reuse data with minimal human intervention [55]. The principles have been endorsed by a diverse set of stakeholders, including academia, industry, funding agencies, and scholarly publishers. The principles emphasize the importance of metadata in making data and digital assets easy to find, with machine-readable metadata being a crucial component of the FAIRification process [55]. In addition to findability, the principles also address the accessibility of data, including the need for authentication and authorization for accessing digital assets [55]. Interoperability is another critical aspect of the principles, with data needing to be integrated with other data and interoperable with applications or workflows for analysis, storage, and processing [55]. The ultimate goal of the FAIR principles is to optimize the reuse of data, and well-described metadata and data are essential for data replication and reuse in different settings [55]. The FAIR principles refer to three types of entities, including data or any digital object, metadata, which is information about the digital object, and infrastructure [55].

F. Open Source

Open source is a development methodology that emphasizes collaboration, transparency, and community involvement [56]. It involves the sharing of software code and the ability to view and modify that code, as opposed to proprietary or closed-source software where the code is not available for viewing or modification [56]. Open source has been a driving force behind many technological advancements, with the internet itself being built on open source software such as Linux [56]. In the field of artificial intelligence, open source development has also played a significant role [21]. Many popular machine learning and deep learning frameworks and libraries, such as Torch, Keras, Hugging Face, and Jax, are open source, allowing for a wider range of use cases, contributions from the community, and access to cutting-edge research [57, 22]. The open-source nature of these frameworks also enables a more transparent and collaborative development process [22].

G. Git

Git is a distributed version control system that is widely used in software engineering as a best practice [58]. It enables developers to track changes in code, collaborate with other developers, and manage different versions of a codebase [58]. The use of Git can improve the efficiency and transparency of software development, making it an essential tool for modern software engineering practices [58]. Git is based on a distributed model, which means that each developer has their own copy of the codebase, and changes can be merged together

with the main codebase through a process of pull requests [58]. This model allows for easy collaboration and review of changes, making it an effective tool for large-scale projects with multiple developers [58].

H. Software Engineering

Software engineering is a well-established discipline that encompasses the process of designing, developing, testing, and maintaining software systems with a focus on quality, reliability, and efficiency [59]. While the specific activities and methodologies involved in software engineering can vary depending on the type of software, the principles of good software engineering practices are generally applicable across all types of software ([59]), including those in the field of artificial intelligence ([12, 15, 32, 60]). The software engineering practices employed in the development of AI systems include, but are not limited to, testing, debugging, documentation, version control, and code review. Additionally, given the complex and evolving nature of AI systems, specific attention must be given to software requirements and their evolution over time ([61, 62]). However, unlike traditional software engineering, the requirements of AI systems may not always be well-defined, and software engineering practices may need to be adapted to the rapidly changing needs of these systems ([61, 62]).

1) Separation of Concerns: The separation of concerns (SoC) is a software engineering principle that suggests that different aspects of a system should be separated into distinct components, allowing for increased clarity, maintainability, and scalability of code ([59, 63]). In the context of AI, this principle can be applied by separating different concerns of a AI into reusable components. By adhering to SoC, researchers can improve the clarity of their code and reduce the risk of introducing bugs and errors ([64, 64, 59, 63]).

2) Modularity: Modularity, or the practice of creating reusable components, is a fundamental aspect of software engineering ([59]). By breaking down complex systems into smaller, reusable components, researchers can improve the understandability and extendability of their code. Additionally, modular code is more easily testable and maintainable, leading to increased reproducibility and reliability of results ([13, 59]).

3) Modualirty in Python: In Python, modular design can be achieved through the use of functions, modules, and libraries ([65]). A python module contains definitions, functions, classes, and variables ([66]). By convention, modules are stored in separate directories, and a directory containing one or more modules is called a package. The presence of `__init__.py` file in a package directory indicates that it is a package, and all files in the directory are considered modules of that package. In other words, the `__init__.py` file makes the directory it's in a Python package, and any code in that file is executed when the package is imported.

I. Developer Experience

The concept of developer experience (DX) is a multidimensional construct that refers to developers' perceptions of various aspects of the development process, including the usability and effectiveness of the tools, frameworks, and platforms used ([67]). DX is a critical factor in software development as it has the potential to impact productivity, motivation, and satisfaction ([67]).

J. Decoupling Concerns in Writing AI Code

Decoupling concerns, also known as SoC, is a crucial design principle in the field of artificial intelligence and machine learning ([64, 68, 59]. For instance, in a typical deep learning experiment, the trainer component is responsible for training the model. The trainer component can be designed to receive either a string representing the dataset/model names or the actual dataset/model objects, with the latter approach providing greater flexibility and customization. Another example of separation of concerns in AI is the data loading and augmentation process, which can be hardcoded into the data loading module or passed as an object to a function. Similarly, a deep learning model can be implemented as a monolithic block of code or as a series of modular components, such as the encoder, decoder, and attention mechanism. The latter approach allows for greater flexibility and customization of the model, as each component can be modified or replaced without affecting the other components.

IV. Related Work

The Yerbamate framework can also be viewed as a project structure design pattern that targets reproducibility and open science. There is currently no standard project structure for data science and machine learning projects. However, there are various practical online project templates[2] and guidelines[3] that aim to provide guidance and organization for these projects. A common trend among these guidelines is towards modularization and separation of concerns, which aims to make projects more manageable and easier to maintain.

A. Coockiecutter

Cookiecutter[4] is a popular tool for setting up project directory structures and boilerplate code for Python-based projects. It provides a predefined directory structure that includes notebooks, reports, figures, references, license, and others to help developers set up a consistent project directory structure illustrated at Figure 1. The Cookiecutter project directory structure separates the src directory into four predefined modules: data, features, models, and visualization, providing a clear separation of concerns. In contrast, Yerbamate allows for the creation of an arbitrary number of independent modules, making it more flexible and easier to manage and reuse code in different projects.

[2]https://medium.com/analytics-vidhya/folder-structure-for-machine-learning-projects-a7e451a8caaa

[3]https://neptune.ai/blog/how-to-organize-deep-learning-projects-best-practices

[4]https://drivendata.github.io/cookiecutter-data-science/

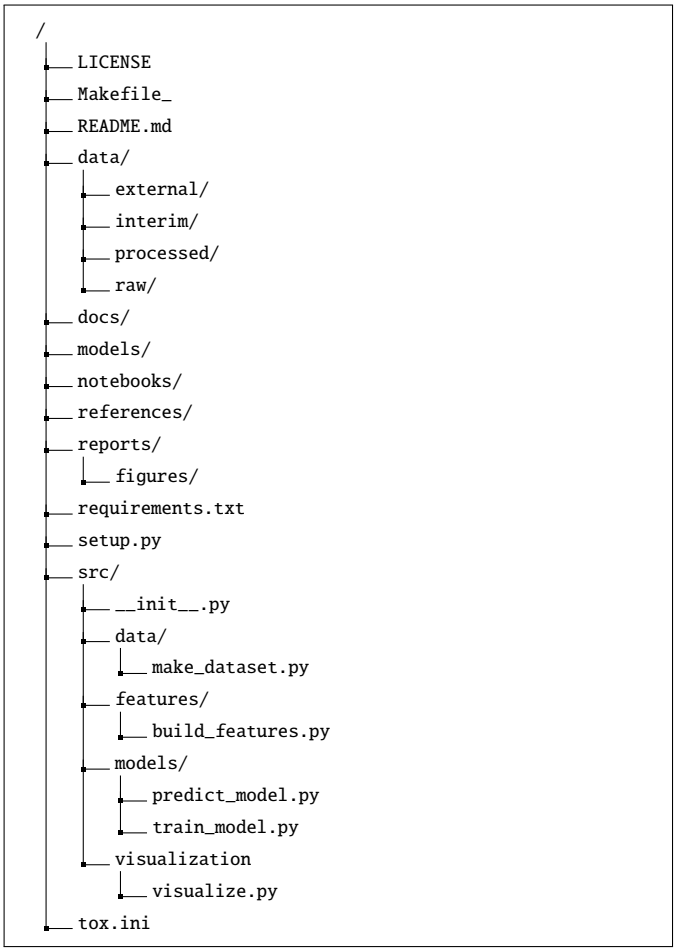

Fig. 1: Cookiecutter's Data Science Directory Tree Structure. The Yerbamate framework can get integrated into the Cookiecutter project structure, managing the src module of the project, and merging the design of license, docs, notebooks, and references.

B. Software Engineering Practices For Accountable AI

Recent research has highlighted the significance of software engineering practices for developing transparent and accountable AI systems. "Towards Accountability for Machine Learning Datasets: Practices from Software Engineering and Infrastructure"[34] proposes a comprehensive guide for documenting the process of dataset development to ensure accountability and mitigate dataset risks. The dataset development lifecycle includes five stages: requirements analysis, design, implementation, testing, and maintenance. The authors recommend creating a comprehensive set of documentation artifacts at each stage, with clearly designated owners and responsibilities to ensure accountability. The documentation aims to create a clear and transparent understanding of the dataset, its development, and its intended uses. The authors also provide a flexible template for documenting dataset requirements that covers a broad range of factors. The proposed documentation guideline can be integrated into the Yerbamate framework.

V. Research Questions

A. What are the most effective software engineering practices for promoting reproducibility and open science in AI research?

Effective software engineering practices for promoting reproducibility and open science in AI and ML research include modularity, separation of concerns, documentation, version control, and testing. Modularity enables the reuse of code and components, while separation of concerns separates code into distinct modules based on their functionality, making it easier to maintain and modify. Documentation helps other researchers understand how to use and reproduce the code, while version control enables tracking of changes to the code over time. Testing helps ensure that the code works as intended and can be used by others. These practices can help promote open science by enabling the sharing of code and promoting transparency in research.

B. How can software engineering principles be utilized for AI to improve the developer experience?

Software engineering principles can be utilized to improve the developer experience in AI development and experimentation by emphasizing modularity, separation of concerns, and standardized naming conventions for modules, such as models, data, and trainers. Standardized project organization, code structure, and modularity enable easier collaboration and sharing among researchers and practitioners. Clear documentation, including contribution guides, tutorials, and support resources, can further facilitate understanding and collaboration, leading to an improved developer experience. Additionally, productivity tools, community driven discussions, open question-answer platforms, and issue tracking systems can be utilized to further enhance collaboration and improve the overall developer experience.

C. How can we enhance open science to promote transparency, reproducibility, and collaboration in scientific research, and foster a more accessible and innovative research environment?

Open science can be enhanced through the development of key tools and technologies that prioritize transparency, reproducibility, and collaboration in scientific research. These tools include open source frameworks, productivity tools for collaboration, open source software and hardware, open-access journals, repositories and data sharing platforms. To make these tools more accessible, it is important to provide clear documentation, training materials, and support resources, as well as promote interdisciplinary collaborations and knowledge sharing. A cultural shift towards open science practices can also be fostered by engaging stakeholders and the public in discussions around the benefits and challenges of AI and open science. By adopting these practices and making these tools more accessible, we can create a more open, transparent, and reproducible scientific community that promotes collaboration and innovation.

VI. Methodology

The methodology section of this study includes the framework design and open science methodology employed in the development of the Yerbamate framework. This study aimed to address the knowledge gap in software engineering for AI by developing a framework that prioritizes open science and improves the developer experience.

A. Open Science

This study introduces and adopts an open source open science methodology that combines both principles to promote transparency, reproducibility, and collaboration in research. The open science methodology employed in this study involves the development of the open source git repository[5] for the Python framework Yerbamate and the creation of an open source open science git repository[6] for Yerbamate. The progress history of both repositories is publicly available, enhancing transparency in the research process and enabling open access to contributions, questions, feedbacks, ideas, and discussions. In addition, the methodology involves disseminating knowledge at different levels of complexity. The Yerbamate framework's documentation[7] and associated tutorial serve as an example of such knowledge dissemination. Furthermore, a Medium article titled "The Ultimate Deep Learning Project Structure: A Software Engineer's Guide into the Land of AI"[8] has been published to disseminate knowledge of software engineering in a more accessible manner. The Yerbamate framework and associated repositories are continuously updated to address limitations, bugs, and improve the developer experience, ensuring that it remains relevant and useful to the research community.

B. Framework Design

The methodology for developing this framework involved analyzing existing open-source python AI projects, reviewing the literature on software engineering practices, and studying modularity, separation of concerns, and open science in AI. The resulting framework prioritizes the creation of independent modules that are standalone and reusable components, which can be used across different projects.

1) Experiment Configuration: Defining hyperparameters and experiments plays a crucial role in AI development and optimization [69]. After evaluating various formats, it was decided that Python is the most suitable format for defining experiments in AI python projects due to its expressiveness, versatility, and straightforward syntax. The integration of existing code and libraries in Python reduces overhead and enhances the experimentation process. Python's powerful libraries and modules enable the exploration of a wider range of hyperparameters and models, leading to a more comprehensive understanding of the problem at hand. The use of well-documented Python code improves the readability of the configuration file, making it more accessible. Additionally, the format is executable directly with Python, which makes it Turing complete, capable of expressing any algorithm, enhancing its flexibility and power.

2) Flexibility and Customization: One of the key challenges in the development of the Yerbamaté framework was ensuring flexibility and customization to meet the varying needs of researchers and practitioners in the field of AI. To address this challenge, Yerbamaté provides increased flexibility by not imposing any restrictions on additional module names, enabling researchers to utilize their preferred module names for custom tasks. Additionally, the framework is designed to be compatible with Python, providing greater flexibility as Python can directly be used to execute experiments and Python files.

C. Evaluation

The Yerbamaté framework was tested on various open-source AI projects across different frameworks and libraries, including Jax, Flax, Pytorch, PytorchLightning, Tensorflow, Keras, and Huggingface, to evaluate its effectiveness. The framework's flexibility was demonstrated through various use cases, such as particle swarm optimization [9][70] and a computer vision image stitching experiment with RANSAC[10] [71]. Furthermore, the framework's effectiveness was evaluated through a case study involving the refactoring of the official implementation of Big Transfer (BiT) to conform to modularity. Code samples, installable modules, examples, and the results of the case study are presented in the Appendix XII.

VII. Results

This section presents the results of the study, which provide software engineering principles for shareable modular Python projects and an open science framework, Yerbamaté. The principles include modular project structuring, independent and interchangeable modules, and an experiment definition format. Furthermore, the Yerbamaté CLI is introduced as a productivity tool for modular Python projects, promoting sharing and reuse of Python modules.

A. Independent/Reusable Python Modules

Independent/Reusable Python modules are self-contained modules that only depend on Python dependencies, such as NumPy, PyTorch, TensorFlow, or Hugging Face. The code inside the module uses relative imports to import within the module, making it independent and reusable once the necessary Python dependencies are installed locally. This approach enhances modularity, reusability, and shareability while promoting good software engineering practices.

B. Separated Concern Python Modular Project Structure

The modular structure of the framework and software engineering best practice of SoC involves organizing the project directory in a hierarchical tree structure, with an arbitrary name

[5]https://github.com/oalee/yerbamate
[6]https://github.com/oalee/os-yerbamate
[7]https://oalee.github.io/yerbamate/
[8]https://medium.com/@alee.rmi/c383f234fd2f
[9]https://github.com/oalee/PSO
[10]https://github.com/oalee/ransac

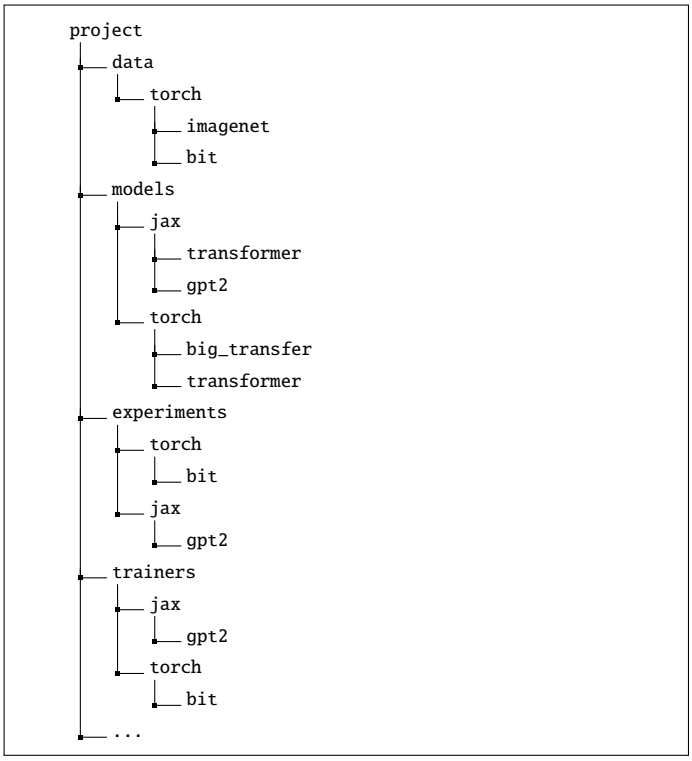

Fig. 2: This figure illustrates a directory tree example of a deep learning modular project. The organization of files into separated concerns in the tree is a common best practice for developing modular Python projects. By adhering to principle of independency, the submodules, such as Transformer and GPT2 can be easily shared and reused across different projects.

given to the root project directory by the user. The project is then broken down into distinct concerns such as models, data, trainers, experiments, analyzers, simulators, each with its own subdirectory. Within each concern, arbitrary modules can be defined, including models, trainers, data loaders, data augmentations, and loss functions. Independent modules can be defined in various heights in the tree.

The framework prioritizes the organization of the project into independent modules when applicable, however there are situations where a combination of independent modules may be necessary for a particular concern. An example of this is the experiment concern, which imports and combines models, data, and trainers to define and create a specific experiment. In such cases, the module is not independent and is designed to combine the previously defined independent modules.

Yerbamaté addresses the issue of specifying the exact framework used for each module in a project that involves multiple frameworks by providing a naming convention that specifies the framework used before the name of the module. This naming convention helps maintain consistency and make it easy to identify which framework was used for a specific module. For instance, the subdirectory for a Jax-based model named "my_model" would be "models/jax/my_model", while the subdirectory for a PyTorch-based model with the same name would be "models/torch/my_model".

C. No-Loop Python Experiment Definition

This section presents a convention and guideline used in the Yerbamate framework for defining Python experiments that prohibits the use of loops. This approach aims to promote a hyperparameter-focused configuration and maintain separation of concerns, resulting in a more flexible and customizable format that can be adapted to various AI tasks, frameworks, and libraries. In practice, the configuration file should primarily consist of hyperparameters, object creations, and executions, while higher complexities such as training loops should generally be avoided.

D. Yerbamaté: A Python Framework for Open Science

Yerbamaté is an open science framework that prioritizes modularity, independent modules, and a separated concern project structure for Python projects. These principles enable Yerbamaté to provide a command line interface (CLI) for easy sharing and installation of code modules, dependency management, and execution of experiments. The framework supports any kind of experimentation with any python framework, with key features such as:

1) Easy Installation with Yerbamaté Install: Yerbamaté's install command enables the installation of modularized projects adhering to separation of concerns principles, along with their associated Python dependencies, with a single command. Additionally, Yerbamaté allows the installation of non-Yerbamaté projects as concrete modules, which can be installed alongside the root module of the project. While Yerbamaté cannot automatically install Python dependencies of this non Yerbamaté repositories, it can for example be used to install source code of over 100 Torch image models or 30 PyTorch vision in transformers directly into a project. See Appendix XII-A.

2) Metadata and Reproducibility: Yerbamaté enhances reproducibility by creating metadata of both python and code dependencies. This metadata complies with the FAIR principles and can be used for exported and reusable modules. By adhering to these principles and guidelines, Yerbamaté promotes FAIR AI, transparency, reproducibility, and open science practices in the research community.

3) Environment API: The Yerbamaté Environment API is a tool provided by the Yerbamate framework to help manage environment variables within an experiment. It is designed to be a central, organized way to set, retrieve, and manage environment-specific information, such as data paths, API keys, database URLs, and other sensitive data. The API prioritizes the use of an `env.json` file for storing environment variables, but can fall back to using the operating system's environment variables if the file is not found.

VIII. Discussion

The decision to use GitHub, a closed-source platform, for an open science project raises concerns regarding the alignment of this platform with the principles of open science. While open source alternatives, such as GitLab and Bitbucket, may be more compatible with open science principles, the authors chose GitHub as a means of facilitating community-based

discussions over git. Despite its closed-source nature, the widespread use of GitHub within the developer community, coupled with its popularity, make it a practical choice for experimenting with contemporary open science methodologies. The authors acknowledge the need for open-source alternatives to GitHub.

Future areas of research and improvement for the Yerbamaté framework include the creation of additional artifacts and tutorials with varying formats and complexity levels, further enhancing knowledge dissemination. The framework's modularity also allows for the addition of more searchable and queryable models, expanding its functionality and providing further utility for researchers. Other potential areas of research and improvement include refining the design, testing, and updating of the framework, as well as addressing identified areas for improvement and refactoring the code.

IX. Social Impact

Machine learning models and artificial intelligence systems have the potential to impact society, both positively and negatively significantly ([72, 73, 74, 25]). Consequently, AI's ethical and societal implications have been the subject of much discussion and research in recent years ([25, 49, 46, 24]). Open science can significantly impact society by promoting collaboration, transparency, and accessibility in research, enabling broader participation in the scientific process and sharing of knowledge and tools, thus accelerating safer progress and ([28, 75, 22, 23, 27, 49]). In artificial intelligence, open science can help democratize access to machine learning and facilitate the development of ethical and accountable AI systems ([49, 47]). By promoting the principles of open science, researchers and practitioners can work together to build more robust, trustworthy, and fair AI solutions ([34, 28, 75, 22, 23, 26, 49].

The development of open source, open science, accessible and user-friendly tools for training machine learning models has the potential to democratize access to artificial intelligence and facilitate more involvement in research and innovation. The Yerbamaté toolkit contributes in this regard, providing a simple and effective means for researchers and practitioners to share, develop and evaluate machine learning models. Moreover, it makes training AI accessible to a broader audience as anyone can run an experiment on accessible science tools such as Colab[11] and reproduce a scientific experiment and conduct their own experiments. The wide accessibility of AI through Yerbamaté and other similar open science tools have the potential to accelerate research in various fields ([19, 18, 76, 20, 17]). At the same time, it is essential to ensure that the development and application of AI adhere to ethical principles, including inclusion, transparency, accountability, and fairness ([34, 73, 25, 75, 74, 72, 49, 24, 46, 48]).

X. Acknowledgments and Contribution Alt−metric

The author acknowledges the invaluable contributions of the open source and open science community, including the developers of tools, libraries, and frameworks such as Jax, Keras, PyTorch, HuggingFace, and many others, as well as researchers and developers who contribute to open source projects. The author expresses gratitude to all the open source code that has been forked and utilized in the development of Yerbamaté. Without these contributions, this work would not be possible. The author acknowledges the contributions of Giulio Zani, the developer of the Maté project. Maté[12] is an AI experimentation framework began development from June 2022 and went underwent various design developments and testing onwards. Due to laws at Maastricht University that prohibit collaborations in research internships, the project was forked into Yerbamaté as an individual effort to enhance flexibility and open science. The author also expresses gratitude to their supervisors, Iris Groen of the University of Amsterdam and Mirela Popa of Maastricht University, for their guidance and support.

The progress and contributions of the Yerbamaté framework development can be tracked using the git commit history as an alt-metric. This enhances the transparency of the research process as the contributions and progress can be analyzed using various tools, including the contribution analyses tool provided on the Yerbamaté repositories[13][14]. Future work could include exploring and refining alternative metrics to more accurately capture community engagement and collaboration, beyond just code contributions. This could involve developing new tools and methods for tracking and analyzing contributions and engagement within the Yerbamate framework and extending it to other open source projects. Additionally, identifying ways to encourage and incentivize community engagement, such as providing rewards or recognition for contributions, could further promote collaboration and improve the development of open source open science framework.

XI. Conclusion

The Yerbamaté framework embodies the principles of FAIR AI and could make a valuable contribution to the field by promoting open science and accessible artificial intelligence. The software engineering principles and modular design approach simplify the development and maintenance of machine learning models, enhance developer experience, and enable collaboration and knowledge sharing among researchers and practitioners with varying technical expertise. The flexibility and customization offered by Yerbamaté enhance its adaptability to meet the diverse needs of AI projects, while its potential to address the reproducibility crisis in AI and facilitate collaborations and sharing makes it a promising tool for future AI research. The promotion of open science frameworks has the potential to revolutionize the development and implementation of trustworthy and transparent AI, fostering innovation and progress across a variety of fields.

[11] https://colab.research.google.com/github/oalee/yerbamate/blob/main/deep_learning.ipynb

[12] https://github.com/ilex-paraguariensis/yerbamate

[13] https://github.com/oalee/yerbamate/graphs/contributors

[14] https://github.com/oalee/os-yerbamate/graphs/contributors

References

[1] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[2] Scott Reed et al. "A Generalist Agent". In: (2022). DOI: 10.48550/ARXIV.2205.06175. URL: https://arxiv.org/abs/2205.06175.

[3] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. DOI: 10.48550/ARXIV.2112.10752. URL: https://arxiv.org/abs/2112.10752.

[4] Andrea Agostinelli et al. *MusicLM: Generating Music From Text*. 2023. DOI: 10.48550/ARXIV.2301.11325. URL: https://arxiv.org/abs/2301.11325.

[5] Kun-Hsing Yu, Andrew L Beam, and Isaac S Kohane. "Artificial intelligence in healthcare". In: *Nature biomedical engineering* 2.10 (2018), pp. 719–731.

[6] Yang Bao, Gilles Hilary, and Bin Ke. "Artificial intelligence and fraud detection". In: *Innovative Technology at the Interface of Finance and Operations: Volume I* (2022), pp. 223–247.

[7] Lijia Chen, Pingping Chen, and Zhijian Lin. "Artificial intelligence in education: A review". In: *Ieee Access* 8 (2020), pp. 75264–75278.

[8] Lasse Espeholt et al. *Skillful Twelve Hour Precipitation Forecasts using Large Context Neural Networks*. 2021. DOI: 10.48550/ARXIV.2111.07470. URL: https://arxiv.org/abs/2111.07470.

[9] Stuart Russell, Daniel Dewey, and Max Tegmark. "Research priorities for robust and beneficial artificial intelligence". In: *Ai Magazine* 36.4 (2015), pp. 105–114.

[10] Francesca Rossi. "Artificial intelligence: Potential benefits and ethical considerations". In: (2016).

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[12] Xufan Zhang et al. *Software Engineering Practice in the Development of Deep Learning Applications*. 2019. DOI: 10.48550/ARXIV.1910.03156. URL: https://arxiv.org/abs/1910.03156.

[13] Saleema Amershi et al. "Software engineering for machine learning: A case study". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE. 2019, pp. 291–300.

[14] Sayash Kapoor and Arvind Narayanan. *Leakage and the Reproducibility Crisis in ML-based Science*. 2022. DOI: 10.48550/ARXIV.2207.07048. URL: https://arxiv.org/abs/2207.07048.

[15] Zhiyuan Wan et al. "How does machine learning change software development practices?" In: *IEEE Transactions on Software Engineering* 47.9 (2019), pp. 1857–1871.

[16] Qinghua Lu et al. "Software engineering for responsible AI: An empirical study and operationalised patterns". In: *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 2022, pp. 241–242.

[17] Zhixi Li, Stuart Keel, and Mingguang He. "Can artificial intelligence make screening faster, more accurate, and more accessible?" In: *The Asia-Pacific Journal of Ophthalmology* 7.6 (2018), pp. 436–441.

[18] Christine T Wolf and Kathryn E Ringland. "Designing accessible, explainable AI (XAI) experiences". In: *ACM SIGACCESS Accessibility and Computing* 125 (2020), pp. 1–1.

[19] Randal S Olson et al. "A system for accessible artificial intelligence". In: *Genetic programming theory and practice XV*. Springer. 2018, pp. 121–134.

[20] Joshua Ong, Seenu M Hariprasad, and Jay Chhablani. "A guide to accessible artificial intelligence and machine learning for the 21st century retina specialist". In: *Ophthalmic Surgery, Lasers and Imaging Retina* 52.7 (2021), pp. 361–365.

[21] Odd Erik Gundersen, Yolanda Gil, and David W Aha. "On reproducible AI: Towards reproducible research, open science, and digital scholarship in AI publications". In: *AI magazine* 39.3 (2018), pp. 56–68.

[22] Gianpaolo Coro. "OPEN SCIENCE AND ARTIFICIAL INTELLIGENCE SUPPORTING BLUE GROWTH." In: *Environmental Engineering & Management Journal (EEMJ)* 19.10 (2020).

[23] Mikio L Braun and Cheng Soon Ong. "Open science in machine learning". In: *Implementing Reproducible Research*. Chapman and Hall/CRC, 2018, pp. 343–365.

[24] Brent Daniel Mittelstadt and Luciano Floridi. "The ethics of big data: current and foreseeable issues in biomedical contexts". In: *The ethics of biomedical big data* (2016), pp. 445–480.

[25] Luciano Floridi et al. "AI4People—an ethical framework for a good AI society: opportunities, risks, principles, and recommendations". In: *Minds and machines* 28 (2018), pp. 689–707.

[26] Daniel J Hicks. "Open science, the replication crisis, and environmental public health". In: *Accountability in Research* (2021), pp. 1–29.

[27] Chris Paton and Shinji Kobayashi. "An open science approach to artificial intelligence in healthcare". In: *Yearbook of medical informatics* 28.01 (2019), pp. 047–051.

[28] Burak Kocak et al. "Transparency in Artificial Intelligence Research: a Systematic Review of Availability Items Related to Open Science in Radiology and Nuclear Medicine". In: *Academic Radiology* (2022).

[29] Victoria Stodden, Peixuan Guo, and Zhaokun Ma. "Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals". In: *PLOS ONE* 8.6 (June 2013), pp. 1–8. DOI: 10.1371/journal.pone.0067111. URL: https://doi.org/10.1371/journal.pone.0067111.

[30] Sacha Epskamp. "Reproducibility and replicability in a fast-paced methodological world". In: *Advances in Methods and Practices in Psychological Science* 2.2 (2019), pp. 145–155.

[31] Elizamary Nascimento et al. *Software engineering for artificial intelligence and machine learning software: A*

systematic literature review. 2020. DOI: 10.48550/ARXIV.2011.03751. URL: https://arxiv.org/abs/2011.03751.

[32] Silverio Martınez-Fernández et al. "Software engineering for AI-based systems: a survey". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.2 (2022), pp. 1–59.

[33] Tommi Mikkonen et al. "Is machine learning software just software: A maintainability view". In: *Software Quality: Future Perspectives on Software Engineering Quality: 13th International Conference, SWQD 2021, Vienna, Austria, January 19–21, 2021, Proceedings 13*. Springer. 2021, pp. 94–105.

[34] Ben Hutchinson et al. *Towards Accountability for Machine Learning Datasets: Practices from Software Engineering and Infrastructure*. 2020. DOI: 10.48550/ARXIV.2010.13561. URL: https://arxiv.org/abs/2010.13561.

[35] Bahar Gezici and Ayça Kolukısa Tarhan. "Systematic literature review on software quality for AI-based software". In: *Empirical Software Engineering* 27.3 (2022), p. 66.

[36] Matthew Hutson. *Artificial intelligence faces reproducibility crisis*. 2018.

[37] D. Sculley et al. "Hidden Technical Debt in Machine Learning Systems." In: *NIPS*. Ed. by Corinna Cortes et al. 2015, pp. 2503–2511. URL: http://dblp.uni-trier.de/db/conf/nips/nips2015.html#SculleyHGDPECYC15.

[38] Samuel J. Bell and Onno P. Kampman. *Perspectives on Machine Learning from Psychology's Reproducibility Crisis*. 2021. DOI: 10.48550/ARXIV.2104.08878. URL: https://arxiv.org/abs/2104.08878.

[39] Sascha Friesike and Thomas Schildhauer. "Open science: many good resolutions, very few incentives, yet". In: *Incentives and performance: Governance of research organizations* (2015), pp. 277–289.

[40] Seokbeom Kwon and Kazuyuki Motohashi. "Incentive or disincentive for research data disclosure? A large-scale empirical analysis and implications for open science policy". In: *International Journal of Information Management* 60 (2021), p. 102371.

[41] Sarah E Ali-Khan, Liam W Harris, and E Richard Gold. "Motivating participation in open science by examining researcher incentives". In: *Elife* 6 (2017), e29319.

[42] Connor O'Carroll et al. "Evaluation of research careers fully acknowledging open science practices-rewards, incentives and/or recognition for researchers practicing open science". In: (2017).

[43] Michael Nielsen. "Reinventing discovery". In: *Reinventing Discovery*. Princeton University Press, 2011.

[44] Jean-Claude Burgelman et al. "Open science, open data, and open scholarship: European policies to make science fit for the twenty-first century". In: *Frontiers in Big Data* 2 (2019), p. 43.

[45] Thijs Devriendt, Pascal Borry, and Mahsa Shabani. "Credit and Recognition for Contributions to Data-Sharing Platforms Among Cohort Holders and Platform Developers in Europe: Interview Study". In: *Journal of Medical Internet Research* 24 (Jan. 2022), e25983. DOI: 10.2196/25983.

[46] Luciano Floridi. "Establishing the rules for building trustworthy AI". In: *Nature Machine Intelligence* 1.6 (2019), pp. 261–262.

[47] Feras A Batarseh and Ruixin Yang. "Data Democracy: At the Nexus of Artificial Intelligence, Software Development, and Knowledge Engineering". In: (2020).

[48] Cathy O'neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown, 2017.

[49] Bryce Goodman and Seth Flaxman. "European Union regulations on algorithmic decision-making and a "right to explanation"". In: *AI magazine* 38.3 (2017), pp. 50–57.

[50] Yu-Tang Hsiao et al. "vTaiwan: An empirical study of open consultation process in Taiwan". In: (2018).

[51] Katrin Praprotnik et al. "Evaluation Report of the Austrian 'Klimarat'UWK, Assessment of the Perspectives of the Members and the Public". In: ().

[52] Connor Shorten and Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning". In: *Journal of big data* 6.1 (2019), pp. 1–48.

[53] Simon Caton and Christian Haas. "Fairness in machine learning: A survey". In: *arXiv preprint arXiv:2010.04053* (2020).

[54] Ninareh Mehrabi et al. "A survey on bias and fairness in machine learning". In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–35.

[55] Mark D Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific data* 3.1 (2016), pp. 1–9.

[56] Chris DiBona and Sam Ockman. *Open sources: Voices from the open source revolution*. " O'Reilly Media, Inc.", 1999.

[57] Giang Nguyen et al. "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey". In: *Artificial Intelligence Review* 52 (2019), pp. 77–124.

[58] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O'Reilly Media, Inc.", 2012.

[59] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill higher education. McGraw-Hill Education, 2010. ISBN: 9780073375977. URL: https://books.google.nl/books?id=y4k%5C_AQAAIAAJ.

[60] Daniel Davis, Jane Burry, and Mark Burry. "Understanding visual scripts: Improving collaboration through modular programming". In: *International Journal of Architectural Computing* 9.4 (2011), pp. 361–375.

[61] Hans-Martin Heyn et al. "Requirement engineering challenges for ai-intense systems development". In: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE. 2021, pp. 89–96.

[62] Hrvoje Belani, Marin Vukovic, and Željka Car. "Requirements engineering challenges in building AI-based complex systems". In: *2019 IEEE 27th Inter-*

national Requirements Engineering Conference Workshops (REW). IEEE. 2019, pp. 252–255.

[63] Bart De Win et al. "On the importance of the separation-of-concerns principle in secure software engineering". In: *Workshop on the Application of Engineering Principles to System Security Design*. Citeseer. 2002, pp. 1–10.

[64] Ran Mo et al. "Decoupling level: a new metric for architectural maintenance complexity". In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 499–510.

[65] Michel F Sanner et al. "Python: a programming language for software integration and development". In: *J Mol Graph Model* 17.1 (1999), pp. 57–61.

[66] Sebastian Raschka. *Python machine learning*. Packt publishing ltd, 2015.

[67] Fabian Fagerholm and Jürgen Münch. "Developer experience: Concept and definition". In: *2012 international conference on software and system process (ICSSP)*. IEEE. 2012, pp. 73–77.

[68] Kai Qian, Jigang Liu, and Frank Tsui. "Decoupling metrics for services composition". In: *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)*. IEEE. 2006, pp. 44–47.

[69] Jia Wu et al. "Hyperparameter optimization for machine learning models based on Bayesian optimization". In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40.

[70] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[71] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60 (2004), pp. 91–110.

[72] Brent Mittelstadt. "Principles alone cannot guarantee ethical AI". In: *Nature machine intelligence* 1.11 (2019), pp. 501–507.

[73] Anna Jobin, Marcello Ienca, and Effy Vayena. "The global landscape of AI ethics guidelines". In: *Nature Machine Intelligence* 1.9 (2019), pp. 389–399.

[74] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[75] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. "Transparent, explainable, and accountable AI for robotics". In: *Science robotics* 2.6 (2017), eaan6080.

[76] Meredith Ringel Morris. "AI and accessibility". In: *Communications of the ACM* 63.6 (2020), pp. 35–37.

[77] Bingchen Liu et al. *Towards Faster and Stabilized GAN Training for High-fidelity Few-shot Image Synthesis*. 2021. DOI: 10.48550/ARXIV.2101.04775. URL: https://arxiv.org/abs/2101.04775.

[78] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[79] Alexander Kolesnikov et al. *Big Transfer (BiT): General Visual Representation Learning*. 2019. DOI: 10.48550/ARXIV.1912.11370. URL: https://arxiv.org/abs/1912.11370.

XII. Appendix

A. Yerbamaté CLI

Yerbamaté CLI provides utility functions and supports modular projects, facilitating machine learning model development. Key CLI options include:

- `mate init module_name`: Initializes a new empty modular project skeleton with the given module name.
- `mate install url -y|n|o pm`: Installs a module from a git repository. Supports multiple formats for the repository URL. The flags -y, -n, and -o specify whether to skip confirmation, skip installing python dependencies, and overwrite existing code modules, respectively. The pm argument specifies the Python package manager to use.
- `mate list`: Lists all available modules in the project.
- `mate exports`: Generates dependencies for reproduciblity and sharing.
- `mate test exp_module exp`: Runs the experiment specified by exp in the module exp_module. Equivalent to `python -m root_module.exp_module.exp test`
- `mate train exp_module exp`: Runs the experiment specified by exp in the module exp_module. Equivalent to `python -m root_module.exp_module.exp train`

1) Example Commands:

- **Installing GAN experiment from a modular project**:
  `mate install oalee/lightweight-gan/lgan/experiments/lgan -yo pip`
- **Training the GAN experiment**:
  `mate train lgan cars`
- **Installing transfer learning experiment**
  `mate install oalee/big_transfer/experiments/bit`
- **Installing +100 models and code from non modular project code**
  `mate install https://github.com/rwightman/pytorch-image-models/tree/main/timm/`
- **Installing +30 Pytorch ViT model source code**
  `mate install https://github.com/lucidrains/vit-pytorch/tree/main/vit_pytorch/`

B. Examples

1) Exported Modules: Yerbamate's mate export command generates a markdown table of reusable components, showcasing metadata such as the module name, type, short URL for installation, exact versions of dependencies for reproducibility, and URLs of code module dependencies. This metadata enhances transparency and reproducibility of research by providing detailed information about each component used in a project. For example, the experiment module may have a dependency on a code module, which is included in the metadata table with its associated information.

name	type	short_url	dependencies
cifar10	data	oalee/deep-vision/deepnet/data/cifar10	['pytorch_lightning~=1.7.5', 'ipdb~=0.13.9', 'torch~=1.12.1', 'torchvision~=0.13.1']
cifar100	data	oalee/deep-vision/deepnet/data/cifar100	['pytorch_lightning~=1.7.5', 'ipdb~=0.13.9', 'torch~=1.12.1', 'torchvision~=0.13.1']
keras	data	oalee/deep-vision/deepnet/data/keras	['tensorflow_gpu~=2.10.0']
timm_aug	data	oalee/deep-vision/deepnet/data/timm_aug	['numpy~=1.24.2']
resnet	experiments	oalee/deep-vision/deepnet/experiments/resnet	['pytorch_lightning~=1.7.5', 'timm~=0.6.12', 'torch~=1.12.1', 'tensorboard~=2.10.0', 'torchvision~=0.13.1', 'https://github.com/oalee/deep-vision/tree/main/deepnet/data/cifar10', 'https://github.com/oalee/deep-vision/tree/main/deepnet/trainers/classification', 'https://github.com/oalee/deep-vision/tree/main/deepnet/models/resnet']
resnet_keras	experiments	oalee/deep-vision/deepnet/experiments/resnet_keras	['keras~=2.10.0', 'https://github.com/oalee/deep-vision/tree/main/deepnet/models/keras/resnet']
kerasnet	experiments	oalee/deep-vision/deepnet/experiments/kerasnet	['ipdb~=0.13.9', 'tensorflow_gpu~=2.10.0', 'keras~=2.10.0']
timm	experiments	oalee/deep-vision/deepnet/experiments/timm	['pytorch_lightning~=1.7.5', 'torch~=1.12.1', 'https://github.com/oalee/deep-vision/tree/main/timm/', 'https://github.com/oalee/deep-vision/tree/main/deepnet/data/cifar10', 'https://github.com/oalee/deep-vision/tree/main/deepnet/trainers/classification', 'https://github.com/oalee/deep-vision/tree/main/deepnet/models/resnet']
resnet	models	oalee/deep-vision/deepnet/models/resnet	['torch~=1.12.1']
keras	models	oalee/deep-vision/deepnet/models/keras	['torch~=1.12.1', 'tensorflow_gpu~=2.10.0']
vit_pytorch	models	oalee/deep-vision/deepnet/models/vit_pytorch	['einops~=0.4.1', 'torch~=1.12.1', 'torchvision~=0.13.1']
torch_vit	models	oalee/deep-vision/deepnet/models/torch_vit	['einops~=0.4.1', 'torch~=1.12.1', 'torchvision~=0.13.1']
classification	trainers	oalee/deep-vision/deepnet/trainers/classification	['pytorch_lightning~=1.7.5', 'torchmetrics~=0.9.3', 'torch~=1.12.1', 'ipdb~=0.13.9']

Fig. 3: Exported module metadata generated from a modular project. The following example is generated from this repository: https://github.com/oalee/deep-vision

2) Example Custom Data Preprocessing: The modular structure of the Yerbamaté toolkit, coupled with its compatibility with pure Python, allows for the integration of custom data preprocessing pipelines with ease. By utilizing the Yerbamaté environment API, developers and researchers can readily access the data paths and results path for the destination of their processed data. For instance, the following project structure illustrated at Figure 4 can utilize the command `python -m deepnet.data.my_data.preprocessing` to execute a custom preprocessing pipeline. The flexibility offered by the python modularity enables users to efficiently tailor their preprocessing procedures to the specific requirements of their research or application, and the Yerbamaté toolkit can be used to share these pipelines effortlessly.

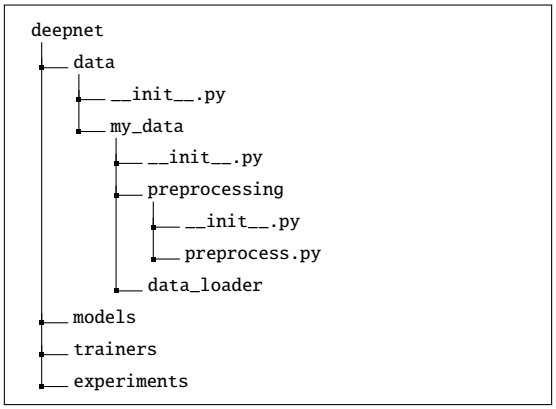

Fig. 4: Custom data preprocessing modular structure example

3) Example Keras Fine Tuning: The following example showcases the flexibility and compatibility of Yerbamaté with the Keras framework. In this experiment definition, the models and trainers do not need to be explicitly defined, as Keras provides these functionalities out of the box. This highlights the ease of integration and adaptability of Yerbamaté with existing frameworks.

```
import os
from tensorflow import keras
from keras.applications.resnet import ResNet50
import yerbamate

env = yerbamate.Environment()
resnet: keras.Model = ResNet50(
    include_top=False,
    input_tensor=keras.Input(shape=(32, 32, 3)),
    classes=10,
    classifier_activation="softmax",
)
resnet.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0004, beta_1=0.9, beta_2=0.999),
    loss="binary_crossentropy",
    metrics=["accuracy", "loss"],
)
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
model_path = os.path.join(env.results, "model.h5")


if env.train:
    resnet.fit(
        x=x_train,
        y=y_train,
        validation_data=(x_test, y_test),
        batch_size=64,
        epochs=10,
    )
    resnet.save(model_path)

if env.test:
    resnet.load_weights(model_path)
    resnet.evaluate(x_test, y_test)
```

4) Example GAN Experiment: The following example illustrates the experiment definition of a Lightweight Generative Adversarial Networks ([77, 78]) implemented with Pytorch Lightning. The use of Python enables the customization of model hyperparameters, loggers, model savers, learning rate schedulers, and optimization algorithms through argument specqification in functions or classes. The source code for the complete project is accessible on Github[15] and all its modules can be installed and the experiment can be trained using Yerbamaté command line on Colab or local machines. The experiment integrates and imports independent trainers, models, and data modules, and defines the experiment's hyperparameters.

```
from ...data.cars import CarsLightningDataModule, AugWrapper
from ...trainers.lgan import LightningGanModule
from ...models.lgan import Generator, Discriminator
from torch import nn
import yerbamate, torch, pytorch_lightning as pl, pytorch_lightning.callbacks as pl_callbacks, os
# Managing environment variables
```

[15]https://github.com/oalee/lightweight-gan

```
env = yerbamate.Environment()

data_module = CarsLightningDataModule(
    image_size=128,
    aug_prob=0.5,
    in_channels=3,
    data_dir=env["data_dir"],
    batch_size=8,
)

generator = Generator(
    image_size=128,
    latent_dim=128,
    fmap_max=256,
    fmap_inverse_coef=12,
    transparent=False,
    greyscale=False,
    attn_res_layers=[],
    freq_chan_attn=False,
    norm_class=nn.BatchNorm2d,
)

discriminator = Discriminator(
    image_size=128,
    fmap_max=256,
    fmap_inverse_coef=12,
    transparent=False,
    greyscale=False,
    disc_output_size=5,
    attn_res_layers=[],  # Try [16, 32, 64, 128, 256] if your hardware allows
)

g_optimizer = torch.optim.Adam(generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
d_optimizer = torch.optim.Adam(
    discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999)
)

model = LightningGanModule(
    save_dir=env["results"],
    sample_interval=100,
    generator=generator,
    discriminator=AugWrapper(discriminator),
    optimizer=[
        {
            "optimizer": g_optimizer,
            "lr_scheduler": {
                "scheduler": torch.optim.lr_scheduler.StepLR(
                    g_optimizer, step_size=100, gamma=0.5
                ),
                "monitor": "fid",
            },
        },
        {
            "optimizer": d_optimizer,
            "lr_scheduler": {
                "scheduler": torch.optim.lr_scheduler.ReduceLROnPlateau(
                    d_optimizer, mode="min", factor=0.5, patience=5, verbose=True
                ),
                "monitor": "fid",
            },
        },
    ],
    aug_types=["translation", "cutout", "color", "offset"],
    aug_prob=0.5,
)

logger = pl.loggers.TensorBoardLogger(env["results"], name=env.name)
callbacks = [
    pl_callbacks.ModelCheckpoint(
        monitor="fid",
        dirpath=env["results"],
        save_top_k=1,
        mode="min",
        save_last=True,
    ),
    pl_callbacks.LearningRateMonitor(logging_interval="step"),
]
```

```
trainer = pl.Trainer(
    logger=logger,
    accelerator="gpu",
    precision=16,
    gradient_clip_val=0.5,
    callbacks=callbacks,
    max_epochs=100,
)

if env.train:
    trainer.fit(model, data_module)
if env.test:
    trainer.test(model, data_module)
if env.restart:
    trainer.fit(model, data_module, ckpt_path=os.path.join(env["results"], "last.ckpt"))
```

XIII. Refactoring Case Study

This section presents a case study on the application of modularity and separation of concerns to the official implementation of "Big Transfer (BiT): General Visual Representation Learning" [79]. The source code from the original repository[16] has been refactored into a modular and decoupled structure in a separate repository[17]. The refactoring process was applied to the PyTorch implementation of the model, and the resulting code can be easily extended and reused. Furthermore, the repository provides the implementation of the model in three popular frameworks: PyTorch, TensorFlow, and JAX, which enhances the accessibility of the work to a wider audience. However, the lack of modularity and separation of concerns in the original implementation could reduce its accessibility and extensibility. Therefore, in this case study, we showcase the process of refactoring the PyTorch code, noting that similar processes could be applied to the TensorFlow and JAX implementations to achieve the same results. The original folder structure of the repository is as following figure:

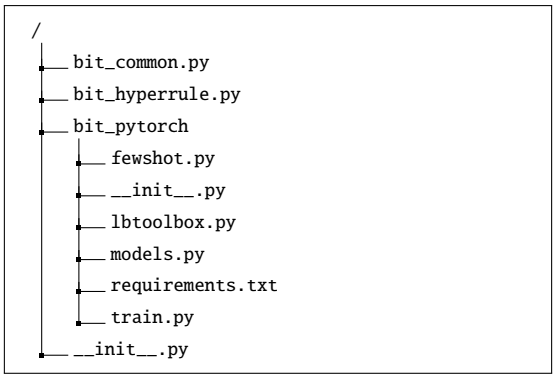

Fig. 5: Official Repository of BiT Project Structure for Pytorch

In this examination, we delve into the modules and Python files of the official Big Transfer repository to better understand the purpose of each one. The components are as follows:

- `bit_common.py` serves as the central point for defining an argument parser and setting up a logger for experiments. Currently, the project only supports a limited set of hyperparameter selections, which include the initial learning rate, batch size, batch split, and five datasets, namely CIFAR10, CIFAR100, Oxford_iiit_pet, Oxford_flowers102, and ImageNet2012. The name of this file, however, does not accurately reflect its purpose.
- `bit_hyperrule.py` is responsible for defining the learning rate scheduler and a utility function for computing the resolution of the model based on the dataset. The name of this file, once again, does not accurately reflect its purpose and separates the concern of data-related functions from the learning rate scheduling.
- `few_shot.py` is specifically designed to find few-shot learning samples for the model, and its name accurately reflects its purpose.
- `lbtoolbox.py` handles interruptions in training and provides a chronometer interface for profiling. This component is independent and does not couple with any other part of the system.
- `models.py` defines the models and is also an independent component that does not couple with any other part.
- `train.py` is utilized for training the model. This component includes the implementation for batch splitting and can only be executed with the pre-defined hyperparameter selection.

[16]https://github.com/google-research/big_transfer
[17]https://github.com/oalee/big_transfer

The following code illustrates the execution of the training procedure in the original repository. The training process is initiated by the `main` function located at the end of the `train.py` file.

```
if __name__ == "__main__":
  parser = bit_common.argparser(models.KNOWN_MODELS.keys())
  parser.add_argument("--datadir", required=True,
                      help="Path to the ImageNet data folder, preprocessed for torchvision.")
  parser.add_argument("--workers", type=int, default=8,
                      help="Number of background threads used to load data.")
  parser.add_argument("--no-save", dest="save", action="store_false")
  main(parser.parse_args())
```

Furthermore, the `main` training function exhibits a high level of coupling between its arguments and objects, as evidenced in the following code snippet of the function definition:

```
def main(args):
  logger = bit_common.setup_logger(args)

  # Lets cuDNN benchmark conv implementations and choose the fastest.
  # Only good if sizes stay the same within the main loop!
  torch.backends.cudnn.benchmark = True

  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  logger.info(f"Going to train on {device}")

  train_set, valid_set, train_loader, valid_loader = mktrainval(args, logger)

  logger.info(f"Loading model from {args.model}.npz")
  model = models.KNOWN_MODELS[args.model](head_size=len(valid_set.classes), zero_head=True)
  model.load_from(np.load(f"{args.model}.npz"))

  logger.info("Moving model onto all GPUs")
  model = torch.nn.DataParallel(model)

  # Optionally resume from a checkpoint.
  # Load it to CPU first as we'll move the model to GPU later.
  # This way, we save a little bit of GPU memory when loading.
  step = 0

  # Note: no weight-decay!
  optim = torch.optim.SGD(model.parameters(), lr=0.003, momentum=0.9)

  # Resume fine-tuning if we find a saved model.
  savename = pjoin(args.logdir, args.name, "bit.pth.tar")
  ....
```

This implementation exhibits limited adaptability as the function is dependent solely on the arguments provided through the command-line interface. The following tree structure and experiment definition showcases modularity and separation of concerns applied on this task.

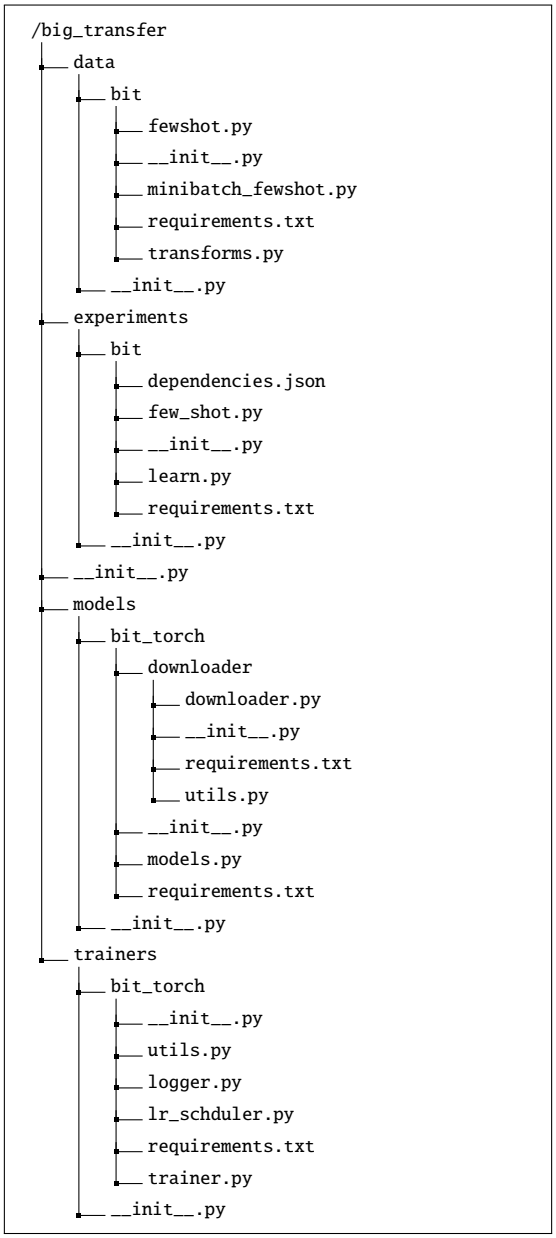

Fig. 6: Refactored Repository of BiT Project Structure

```
from ...trainers.bit_torch.trainer import test, train
from ...models.bit_torch.models import load_trained_model, get_model_list
from ...data.bit import get_transforms, mini_batch_fewshot
import torchvision as tv, yerbamate, os, tensorboard
from torch.utils.tensorboard import SummaryWriter


# BigTransfer Medium ResNet50 Width 1
model_name = "BiT-M-R50x1"
# Choose a model form get_model_list that can fit in to your memoery
# Try "BiT-S-R50x1" if this doesn't works for you

env = yerbamate.Environment()

train_transform, val_transform = get_transforms(img_size=[32, 32])
data_set = tv.datasets.CIFAR10(
    env["datadir"], train=True, download=True, transform=train_transform
)
val_set = tv.datasets.CIFAR10(env["datadir"], train=False, transform=val_transform)

train_set, val_set, train_loader, val_loader = mini_batch_fewshot(
    train_set=data_set,
```

```
    valid_set=val_set,
    examples_per_class=None,  # Fewshot disabled
    batch=128,
    batch_split=2,
    workers=os.cpu_count(),  # Auto-val to cpu count
)

imagenet_weight_path = os.path.join(env["weights_path"], f"{model_name}.npz")
model = load_trained_model(
    weight_path=imagenet_weight_path, model_name=model_name, num_classes=10
)
logger = SummaryWriter(log_dir=env["results"], comment=env.name)

if env.train:
    train(
        model=model,
        train_loader=train_loader,
        valid_loader=val_loader,
        train_set_size=len(train_set),
        save=True,
        save_path=os.path.join(env["results"], f"trained_{model_name}.pt"),
        batch_split=2,
        base_lr=0.001,
        eval_every=100,
        log_path=os.path.join(env["results"], "log.txt"),
        tensorboardlogger=logger,
    )

if env.test:
    test(
        model=model,
        val_loader=val_loader,
        save_path=os.path.join(env["results"], f"trained_{model_name}.pt"),
        log_path=os.path.join(env["results"], "log.txt"),
        tensorboardlogger=logger,
    )
```

The new structure in the refactored repository depicted at Figure 6 is designed to address the limitations of the original implementation. By separating concerns and adopting modularization, the refactored repository provides a more flexible and scalable solution for training models. The different components in the new structure, such as the trainer, model, and data loading modules, are designed to be independent and reusable, making it easier to manage and maintain the codebase. Moreover, the experimentation module provides a unified interface for combining and executing the various components, making it easier to experiment with different configurations and hyperparameters. Overall, the new structure in the refactored repository represents a significant improvement over the original implementation, offering a better and more organized approach to training machine learning models.