Training future ML engineers: a project-based course on MLOps

Filippo Lanubile, University of Bari, Bari, Italy

Silverio Martínez-Fernández, Universitat Politècnica de Catalunya, Barcelona, Spain

Luigi Quaranta, University of Bari, Bari, Italy

***Abstract*— Recently, the proliferation of commercial ML-based services has given rise to new job roles, such as ML engineers. Despite being highly sought-after in the job market, ML engineers are difficult to recruit, possibly due to the lack of specialized academic curricula for this position at universities. To address this gap, in the past two years, we have supplemented traditional Computer Science and Data Science university courses with a project-based course on MLOps focused on the fundamental skills required of ML engineers. In this paper, we present an overview of the course by showcasing a couple of sample projects developed by our students. Additionally, we share the lessons learned from offering the course at two different institutions.**

***Keywords*— machine learning, data science, software engineering for AI**

As machine learning (ML)-based systems become increasingly complex, there is a growing demand from the industry for ML engineers, also known as "AI engineers" [1]. ML engineers are professionals – trained in both SE and ML – who can handle the end-to-end process of building and maintaining production-ready ML components; to accomplish this, they leverage a varied array of practices and tools generally recognized under the umbrella term of MLOps.

Despite the growing job market demand, at present, there is almost no offer of specialized academic curricula for ML engineers.[1] Indeed, most AI university programs – mainly concerned with teaching state-of-the-art ML techniques – miss the opportunity to train future ML specialists on building production-grade components out of ML prototypes. For this reason, aspiring ML engineers need to look beyond university programs to learn the new craft and, indeed, several specialized online courses have been published lately, compensating for the absence of academic options.

Even so, it is still challenging for companies to recruit qualified ML engineers, with costly repercussions on their ability to bring ML products to the market. For instance, in a recent survey conducted by Algorithmia, many firms acknowledged facing significant challenges in the deployment of their models, despite the substantial effort and significant investments [2]. Christian Kästner – author of the book "Machine Learning in Production" [3] and lecturer of the homonymous course at Carnegie Mellon – argues that *"'software engineering for ML' is more of an education problem than a research problem"* [4]: most blocking challenges that practitioners experience in the field would be solved by empowering data scientists with software engineering knowledge. In the same line of thought, as software engineering educators, we believe there is a pressing need to train ML specialists on SE best practices and tools. Hence – looking forward to having a specialized degree program on ML engineering offered at our universities – we have started exploring the feasibility and outcomes of teaching MLOps fundamentals in a 3-month university course.

[1] With a few notable exceptions, e.g., the Master's of AI Engineering offered at Carnegie Mellon (CMU).

[SIDEBAR] INDUSTRIAL TRENDS ON MLOPS

ML engineers have been actively participating in MLOps communities, fostering mutual support as they tackle work challenges together. They have also contributed crowd-sourced lists of MLOps resources and tools, like "Awesome MLOps"[2]. "MLOps Community"[3] is a prominent example of an online hub where MLOps professionals gather to share their practical experiences, learn new skills, and collaborate on projects. Visiting the community website is a great way to stay updated on the latest trends in MLOps. For instance, at present, the community is primarily focused on the challenge of managing Large Language Models (LLMs) in production.

Regarding MLOps tools, the latest trends can be also inferred by checking curated lists like MLOps.toys.[4] Notably, several of the tools available there are publicly contributed on GitHub, which indicates an increasing and needed involvement of the open-source community in this space.

DESIGNING A PROJECT-BASED COURSE ON MLOPS

In 2021, we decided to explore if our students from the Master's program on Computer Science at the University of Bari, Italy (hereafter "Uniba") and the Bachelor's program on Data Science at the Universitat Politècnica de Catalunya, Barcelona, Spain (hereafter "UPC") could successfully build and deploy ML-based components while training on MLOps fundamentals. With this goal in mind, we designed a project-based course on MLOps [7], focused on the demonstration and hands-on experience with MLOps solutions for the end-to-end development of ML-based components. We offered the first edition of the course at Uniba in Fall 2021. Upon collecting encouraging feedback from the students, in Fall 2022, we offered a second edition at both Uniba and UPC. During this second edition, we decided to collect data from the students to evaluate our course, answering the following questions:

Q1: How well does our MLOps course align with student expectations?

Q2: How do students perceive the usefulness of the content and teaching methodology employed in our course?

Course evaluation

As general evidence of course effectiveness, we considered the ability of the students to carry out their project activities end-to-end, meeting all deadlines.

To measure how the students perceived the benefits and, more in general, the experience of project-based learning of MLOps, we conducted a survey-based feedback study. Specifically, we administered a first survey at the beginning of the course – to gauge the students' initial knowledge and expectations – and a second survey at project delivery – to evaluate final impressions.[5]

We summarized quantitative answers with descriptive statistics and open-ended ones with thematic analysis. In the latter case, we adopted a focused coding approach: after familiarizing ourselves with the collected data, we defined a tentative set of codes. Then, we selectively coded relevant segments of text, constantly refining the initial codes and taking note of the most interesting excerpts. Finally, we grouped codes into themes and reviewed the analysis results with the whole team.

Course design

In both editions of the course, we focused on six core skills typically expected from ML engineers:

1. scoping a real-world ML problem and coordinating teamwork;
2. ensuring ML pipeline reproducibility;
3. fostering quality assurance (QA);
4. developing an API for ML;
5. delivering an ML component;
6. keeping the feedback loop.

[2] https://github.com/visenger/awesome-mlops

[3] https://home.mlops.community/

[4] https://mlops.toys

[5] The two surveys used in this study are available at: 10.5281/zenodo.8026803.

We asked our students to work in teams of 3-5 people to turn a prototypical ML model into a production-ready ML component. Here, by “production-ready”, we mean a component that can be easily integrated into a production-grade system and effortlessly maintained over time. As such, we expect it to:

- be the product of a *reproducible* build process that can be fully automated with CI/CD tools;
- have *production-grade quality*, i.e., to be properly tested and checked with QA tools;
- expose a *cross-platform* API and be packaged in a portable way.

One of the challenges we faced while designing the course was the selection of tools to exemplify MLOps implementation. Not only the related practices are still consolidating and far from being standardized, but also the multitude of available MLOps tools keeps evolving at a stunningly fast pace (see Figure 1). To reach our final selection, for each MLOps practice, we considered the following criteria: our picks had to be (1) preferably open source (2) popular in the MLOps community, (3) well-documented, and (4) easy to learn. We left our students free to explore other options anyway and make their own informed decisions, regardless of our choices.

We organized the course and projects into six milestones, corresponding to the ML engineering skills above. In the following paragraphs, we will go through each skill, motivating its importance and showing how a couple of student teams applied the related practices in their work. Their projects are just representative examples of several other projects developed by our students. Being based on particular ML models – freely selected by the teams at the course start – each project posed specific challenges and inspired distinct solutions. Often, the students went beyond our demonstrations, adopting additional or alternative tools to meet their specific project requirements. For the benefit of all, we asked each team to report their experience to the class in bi-weekly retrospective meetings.

Scoping an ML Problem and Coordinating Teamwork

At the beginning of the course, we asked all teams to set up communication and collaboration platforms to coordinate their work. Then, we tasked them with scoping a real-world problem to be solved with ML and selecting (or building) a prototypical model.

Effective communication is crucial in collaborative software development. Defining clear guidelines in this regard helps team members stay consistent in how they share information, for the benefit of team awareness and information retrieval. In class, we demonstrated the use of Microsoft Teams[6] (Uniba) and Slack[7] (UPC), for synchronous communication, while for project coordination, we demoed a Kanban-style board using Trello.[8]

Once all teams had selected or built their model, we demonstrated how to document it using “model cards”. This lightweight approach to model specification, originally proposed by Mitchel et al. [8] and lately popularized by Hugging Face[9], consists of templates providing a structured description of models, including the ML algorithm, training dataset, and use cases. A similar approach can be employed for the specification of datasets.[10] We chose this solution because, with sustainable effort, it allows for concise reporting of all relevant aspects of an ML project.

Example projects

Here we briefly introduce a couple of sample projects – as reported in the corresponding model cards – that will be referenced throughout this article.

“Math Symbol CNN” (hereafter “MS”), a team of students from Uniba, built a computer vision (CV) system for the classification of mathematical symbols (e.g., +, sin, and log) in low-resolution images (i.e., 28x28 pixels) of handwritten text. To this aim, they leveraged the refined version of a convolutional neural network built for another course.

“Crystal Gazers” (hereafter “CG”), a team of students from UPC, built a natural language processing (NLP) application aimed at predicting the omitted word in a sentence based on the context provided by surrounding words. The students employed a transformer model trained from scratch on a dataset of Wikipedia articles in Catalan.

Ensuring ML Pipeline Reproducibility

Reproducibility is a key requirement in ML projects: not only it is important to get consistent performances – in production as in the lab – but also to enable the recovery and timely retraining of deployed models. However, achieving reproducibility in ML is challenging. We

[6] https://www.microsoft.com/en-us/microsoft-teams/group-chat-software
[7] https://slack.com
[8] https://trello.com
[9] https://huggingface.co/docs/hub/model-cards
[10] https://huggingface.co/docs/hub/datasets-cards

address this topic in our second course milestone, aimed at providing students with the knowledge and skills required to build reproducible ML pipelines.

A first step towards reproducibility is embracing version control. Concerning code, in class, we exemplified the use of git – the de-facto standard version control system – with GitHub, the most popular platform for git repository hosting. Also, we recommended using the GitHub flow [9], a lightweight, branch-based workflow for collaborative software development. Despite knowing git, several students admitted to not using version control in data science projects. For instance, they would normally ignore Jupyter notebooks as related diffs are hard to read. We emphasized the importance of versioning all code artifacts and recommended using modern editors or specialized Jupyter extensions, like nbdime, for improved notebook diff display.

On the other hand, versioning data is more challenging than code. Different data formats (e.g., text, images) require specialized versioning mechanisms; moreover, storing and retrieving data is harder due to the larger file sizes. In class, we showed how these challenges can be overcome using specialized tools; a popular example is DVC[11], an open-source platform used to version large data files (datasets and models) and back them up to cloud remotes.

Experiment tracking is another reproducibility keystone. Being able to trace back experimental decisions is crucial to identify and reproduce the best experimental paths. To support this practice, we demonstrated MLflow Tracking[12], a popular open-source solution. Besides the Tracking module, the "MS" team leveraged MLflow's Registry module to save models in a centralized store; moreover, they employed DagsHub[13] – a cloud hosting platform for data science projects – as a remote for both DVC and MLflow. Other teams preferred tracking their experiments with Tensorboard, mainly because of its tight integration with Tensorflow.

Fostering QA

Previous research has found that the quality of code in experimental ML artifacts is generally poor, especially in the case of computational notebooks [10], [11]. Similarly, model performance is known to be largely affected by the quality of training data, which is far from ideal in real-world scenarios. Our third course milestone focuses on QA, aiming to provide the students with practical guidance for quality improvement.

To ensure production-grade quality for artifacts developed in the lab, data scientists need to modularize, test their code, and check it with static analyzers. Our students straightforwardly incorporated the recommended QA tools into their pipelines. For instance, after consolidating experimental notebooks into a pipeline of Python scripts, the "MS" team used *Pylint* to statically analyze their code and a combination of *Pytest* and *unittest* to test it. In addition, "CG" checked their repository with Pynblint [12] – a specialized static analyzer for Jupyter notebooks. Besides, to optimize energy efficiency, they tracked the CO2 emissions of their pipelines using Code Carbon.

As versioning, quality assurance is more challenging for data than for code. Due to the variety of existing data formats, there is no tool covering all possibilities. In class, we demonstrated Great Expectations (GE), an open-source framework allowing the definition of assertions on various properties of tabular data. However, since none of the teams had trained their model on tabular data, it was challenging for them to find workarounds. Some students resorted to testing only preprocessed data with GE (e.g., "CG" used GE to check if tokens extracted from a text were all integers). In contrast, other teams preferred using alternative solutions (e.g., the "MS" team used *Deepchecks* for its native support of image data).

Concerning model QA, we showed how to complement the use of quantitative metrics, like precision and recall, with behavioral model testing. Behavioral tests assess the behavior of models when applied to specific categories of input data. In class, we exemplified them in the NLP domain, as inspired by [13]. Interestingly, some teams like "MS" showed how the same idea can be applied to different domains, like CV.

API Development for ML

To enable their seamless integration into larger systems, ML models typically expose their predictive capabilities through web APIs. We devote the fourth milestone of our course to showing how to wrap ML models with REST APIs using FastAPI. We selected this particular framework for its shallow learning curve, but also for its compliance with the OpenAPI standard.

Most teams followed our recommendation and adopted FastAPI. Conversely, a few groups resorted to alternative

[11] https://dvc.org
[12] https://mlflow.org
[13] https://dagshub.com/about

solutions; for instance, "CG" used AWS API Gateway – an AWS-managed service for web API development – to expose HTTP endpoints for their model. The students could also build a demo application to demonstrate their API. Despite not being trained in web development, several of them could build a client web app in no time with special-purpose front-end frameworks like Gradio (e.g., "MS") and Streamlit (e.g., "CG").

Component Delivery

Another crucial set of skills required of ML engineers concerns the delivery of ML-based components. Beyond exposing endpoints, models need to be packaged in a portable way and automatically deployed in cloud-based production environments. In our fifth course milestone, we show how to achieve this using containerization and CI/CD technologies.

Being the de-facto standard, we used Docker to exemplify software containerization. The students found it relatively easy to understand and apply. For instance, "MS" straightforwardly employed *Docker* and *Docker Compose* to implement a 4-components microservices architecture. However, some students had a hard time setting up containers for models requiring a GPU at inference time. Some of them identified suitable base images to leverage full model performance with GPUs, while others packaged a simplified version of their model as a workaround.

Next, we showed how to automate the whole build and deployment process with CI/CD tools. Due to its seamless integration with GitHub, we used GitHub Actions to demonstrate this practice. However, some teams preferred using the facilities offered by their cloud provider. For instance, "MS" deployed their multi-container system to Okteto and leveraged its native support for Docker Compose builds; differently, "CG" employed AWS facilities to run their components in EC2 instances.

Keeping the Feedback Loop

To ensure service availability and performance after deployment, it is crucial to continuously monitor ML-enabled components. A monitoring system should track both the resource consumption of ML components as well as the performance of ML models themselves, as they are typically subject to performance degradation over time. By setting up a monitoring system, ML engineers ensure to keep the feedback loop, being able to timely replace their models as needed. Hence, we dedicated the final milestone of our course to monitoring practices for ML-based systems.

All teams were able to set up a monitoring system for their ML-based component. They mostly followed the examples provided in class, based on two popular open-source solutions often used in tandem to track system metrics (Prometheus) and visualize them in a dashboard (Grafana).

LESSONS LEARNED

All teams could successfully turn their model prototype into a production-ready ML component. Also, they all coped well with the project deadlines and managed to deploy their product to the cloud. In the following paragraphs, we will examine the feedback collected from the students. The lessons learned by analyzing their course experience will help us improve the next course editions.

Expectations of the students

At the beginning of the course, we assessed the prior knowledge and learning expectations of our students. We conducted an anonymous survey, collecting 51 responses (23 at Uniba and 28 at UPC).

To start, we asked each student to self-report their experience in both SE and ML using a 5-point Likert scale, where 1 represents a "very poor" experience and 5 an "excellent" experience. Consistently across Uniba and UPC – the students exhibited greater confidence in ML, with 86.28% reporting an "average" or "above average" experience. Specifically, almost all of them (94.12%) had had previous experience with CV and most of them with NLP (e.g., 78.43% had worked on text classification). Conversely, only 58.8% of the students indicated an "average"/"above average" experience in SE.

Then, we checked what the students were expecting to learn from the course. More than half of them (29) anticipated learning *"engineering practices to build production-ready ML-based systems"*. Eighteen of them mentioned the application of specific software engineering best practices to ML (e.g., versioning or containerization):

> *"I am interested in understanding containerization, which is currently a very popular solution that I have never had the opportunity (and the time) to experiment with."*

Eight students expected training on top-notch technologies to support the building process of ML-enabled systems; differently, five of them anticipated learning engineering best practices for better management of data science projects (9):

> *"I expect to learn how to improve the way an ML project is carried out from the beginning to the end, through the different stages."*

Five students expected to extend their knowledge of software engineering; conversely, six of them thought they were going to learn more about machine learning. Finally, only three students mentioned expecting to know more about the best practices for collaboration in ML projects and just a couple about big data management and privacy.

Overall, our course was able to meet by design most of the student expectations, the main exceptions being extended knowledge about ML, and big data management and privacy.

Perceived usefulness of the course contents and teaching methodology

By the end of the course – and before the final exam – we asked the students to provide final feedback. Once again, we administered an anonymous survey; this time we collected 44 responses in total (18 at Uniba and 26 at UPC).

To begin, we assessed student agreement on the usefulness of the MLOps practices presented in class using a 5-point Likert scale ranging from "Strongly disagree" (1) to "Strongly agree" (5). The most "strongly agreed" practices were code versioning (83.33%), API design for ML (61.90%), and experiment tracking (59.09%). Likewise, we surveyed student opinions about our teaching methodology. Most of them found the project-based nature of the course (63.64% "Strongly agree", 29.55% "Agree") and teamwork (52.27% "Strongly agree", 29.55% "Agree") helpful to learn. 84.09% of the students considered the project feasible, and 90,91% agreed or strongly agreed about the appropriateness of the project milestones. Conversely, 15.91% of the students were neutral about the appropriateness of workload distribution, while 22.73% disagreed or strongly disagreed.

These encouraging results were confirmed by our manual analysis of the open-ended answers. We learned that most of the students (29) found the course useful and were willing to reuse some or most of the proposed practices and tools in their future projects.

> *"I knew about some of these practices before, but never actually implemented them. Having to do so was useful and taught me a lot for future projects."*

A couple of them claimed they had already started to do so by the end of the course.

> *"I have already started applying what we have learned during this course to other ML projects. This kind of practice has solved a lot of problems that I encountered while developing ML models over the past year."*

Eight students highlighted specific tools or categories thereof they found particularly useful while developing their project, e.g.: collaborative versioning with git and GitHub (4), experiments tracking with MLflow (3), the Cookiecutter project structure (2), or building data pipelines with DVC. A couple were willing to reuse especially the tools offering support for reproducibility.

Finally, five students reported learning the advantages of using SE best practices when building ML-based systems. All in all, these results show that students already acquainted with ML enjoy learning state-of-the-art engineering practices and tools to improve their ML workflow.

Suggestions for improvement

Seven students expressed criticisms about the course or recommended changes for future editions. For instance, a couple of them would have appreciated more guidance on the use of Git (and GitHub) or the deployment of an ML-based component to the cloud.

Three students complained about the workload of the course, which they found too heavy:

> *"I enjoyed the course, although it has taken most of my study time."*

or about the general overhead of applying software engineering practices to ML projects:

> *"All of this is important and the subject has made me realize it. However, applying these practices doubles the time spent to develop a project."*

Besides, a couple of students reported not being happy with some of the recommended tools; in particular, they complained about Great Expectation, either because its use is redundant in their project (*"[Great Expectations] is used to ensure data quality standards, but by preprocessing data before using it for training we already ensure them."*) or because it does not scale to larger projects (*"several of these tools are incomplete, in the sense that they can only be used in relatively small projects; for example, Great Expectations…"*).

Appreciation for the course

Finally, we examined the willingness of the students to recommend our course to their colleagues using a 4-point Likert scale ranging from "Definitely not" to "Definitely." Most of the students declared to be likely (36.36%) or definitely likely (54.55%) to promote the class. Eleven of them also expressed explicit appreciation for the course in the open-ended items of the survey, confirming it has become a necessary addition to the traditional academic curriculums:

"Nowadays, ML-based systems are everywhere, and it is necessary to have this course. It would be great if it could be extended into a 9-credit course."

"I found it really useful. I think having this type of subject in our degree is crucial. I have used and I will use what I have learned."

CONCLUSION

In this article, we shared our experience in designing and delivering a project-based university course on MLOps aimed at training future ML engineers. After examining the practices addressed and a selection of tools used by our students to overcome engineering challenges in the development of their project works, we shared their feedback on the course. From our experience, we learned that students already acquainted with ML are eager to know more about engineering best practices for ML and that core competencies required of ML engineers can be successfully taught over the course of a semester.

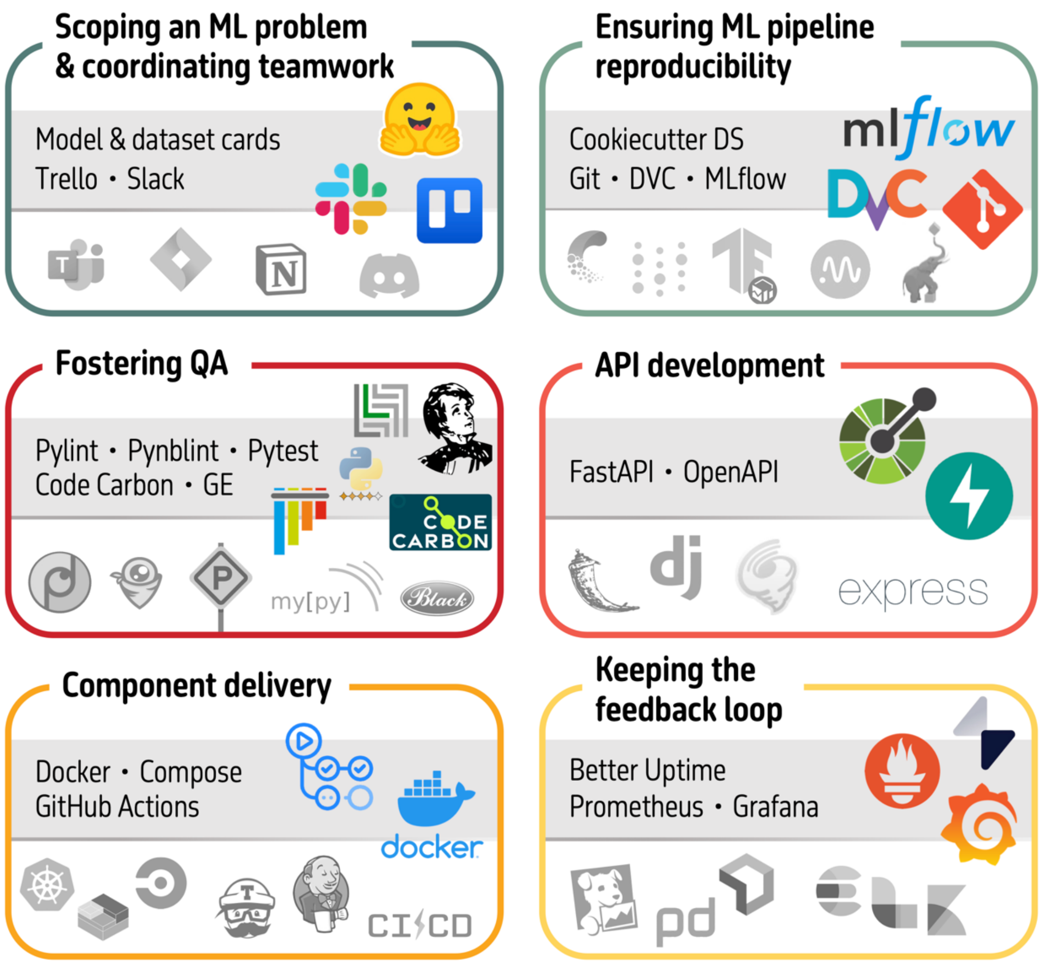

FIGURE 1. This figure roughly depicts the vast technology landscape of MLOps. We group a small sample of the existing tools by the skills addressed in our course; for each skill, our picks are highlighted in color, while possible alternatives are in greyscale.

ACKNOWLEDGMENTS

This work is partially supported by the NRRP Initiative – Next Generation EU ("FAIR - Future Artificial Intelligence Research", code PE00000013, CUP H97G22000210007); the Complementary National Plan PNC-I.1 ("DARE - DigitAl lifelong pRevEntion initiative", code PNC0000002, CUP B53C22006420001), and the project TED2021- 130923B-I00, funded by MCIN/AEI/10.13039/50110001 1033 and the European Union Next Generation EU/PRTR.

REFERENCES

[1] I. Ozkaya, “An AI Engineer Versus a Software Engineer,” *IEEE Softw.*, vol. 39, no. 6, pp. 4–7, Nov. 2022, doi: 10.1109/MS.2022.3161756.

[2] Algorithmia, “2020 state of enterprise machine learning,” 2020.

[3] C. Kästner, *Machine Learning in Production*. 2021. Accessed: Jan. 31, 2023. [Online]. Available: https://ckaestne.medium.com/machine-learning-in-production-book-overview-63be62393581

[4] *MSR'22 Keynote: From Models to Systems: Rethinking the Role of Software Engineering for ML*, (May 20, 2022). Accessed: Feb. 09, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=_m-m90S_4Gg

[5] D. Sato, A. Wider, and C. Windheuser, "Continuous Delivery for Machine Learning - Automating the end-to-end lifecycle of Machine Learning applications," Sep. 19, 2019. https://martinfowler.com/articles/cd4ml.html

[6] Q. Lu, L. Zhu, X. Xu, Z. Xing, and J. Whittle, "Towards Responsible AI in the Era of ChatGPT: A Reference Architecture for Designing Foundation Model-based AI Systems." arXiv, May 23, 2023. Accessed: Jun. 05, 2023. [Online]. Available: http://arxiv.org/abs/2304.11090

[7] F. Lanubile, S. Martínez-Fernández, and L. Quaranta, "Teaching MLOps in Higher Education through Project-Based Learning." arXiv, Feb. 02, 2023. Accessed: Feb. 09, 2023. [Online]. Available: http://arxiv.org/abs/2302.01048

[8] M. Mitchell *et al.*, "Model Cards for Model Reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, Atlanta GA USA: ACM, Jan. 2019, pp. 220–229. doi: 10.1145/3287560.3287596.

[9] GitHub, "GitHub flow," *GitHub Docs*. https://docs.github.com/en/get-started/quickstart/github-flow (accessed Jan. 31, 2023).

[10] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks," in *Proc. of the 16th International Conference on Mining Software Repositories*, 2019, pp. 507–517. doi: 10.1109/MSR.2019.00077.

[11] J. Wang, L. Li, and A. Zeller, "Better code, better sharing: On the need of analyzing jupyter notebooks," in *Proc. of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ACM, 2020, pp. 53–56. doi: 10.1145/3377816.3381724.

[12] L. Quaranta, F. Calefato, and F. Lanubile, "Pynblint: a Static Analyzer for Python Jupyter Notebooks." May 24, 2022. doi: 10.1145/3522664.3528612.

[13] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond Accuracy: Behavioral Testing of NLP models with CheckList," *Association for Computational Linguistics*, 2020, doi: 10.48550/ARXIV.2005.04118.

FILIPPO LANUBILE is a full professor of Computer Science and head of the Department of Informatics at the University of Bari, Italy, where he also leads the Collaborative Development Research Group. His research interests include human factors in software engineering, collaborative software development, software engineering for AI/ML systems, social computing, and emotion detection. Contact him at filippo.lanubile@uniba.it.

SILVERIO MARTÍNEZ-FERNÁNDEZ is an assistant professor at the Universitat Politècnica de Catalunya (UPC)-BarcelonaTech. His research interests include empirical software engineering, software engineering for AI/ML-based systems, and green AI.

Contact him at silverio.martinez@upc.edu.

LUIGI QUARANTA is a research fellow in the Collaborative Development Research Group at the University of Bari, Italy. His research interests include software engineering for AI/ML-based systems, MLOps, and computational notebooks. Contact him at luigi.quaranta@uniba.it