Continuous Machine Learning

Continuous Learning (Image by Author)

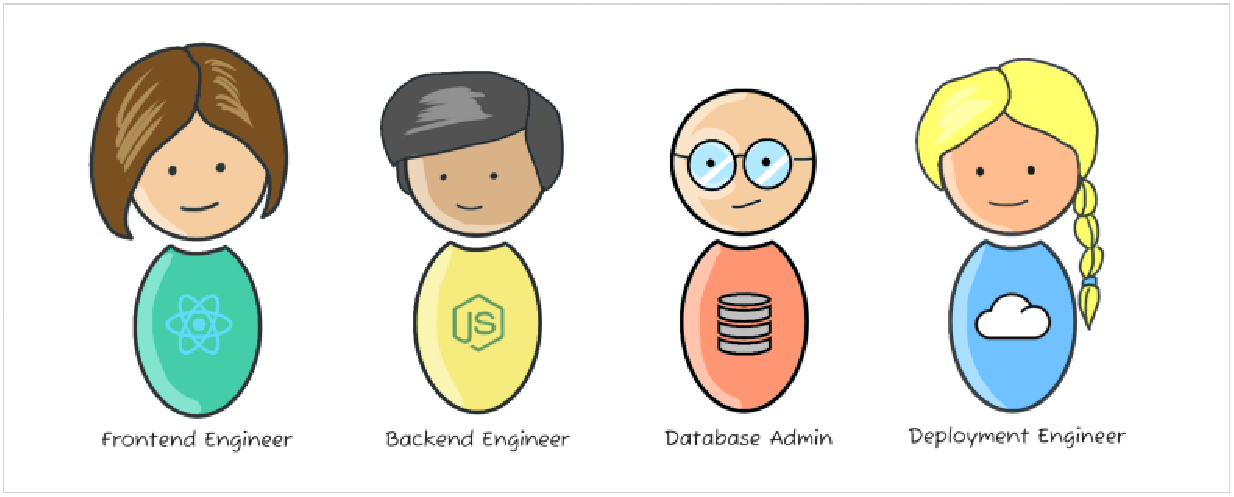
An Introduction to CML (Iterative.ai)

This article is for data scientists and engineers looking for a brief guide on understanding Continuous Machine Learning, What it is? How does it work? and how you can integrate it into your daily model development process?

Let us begin!

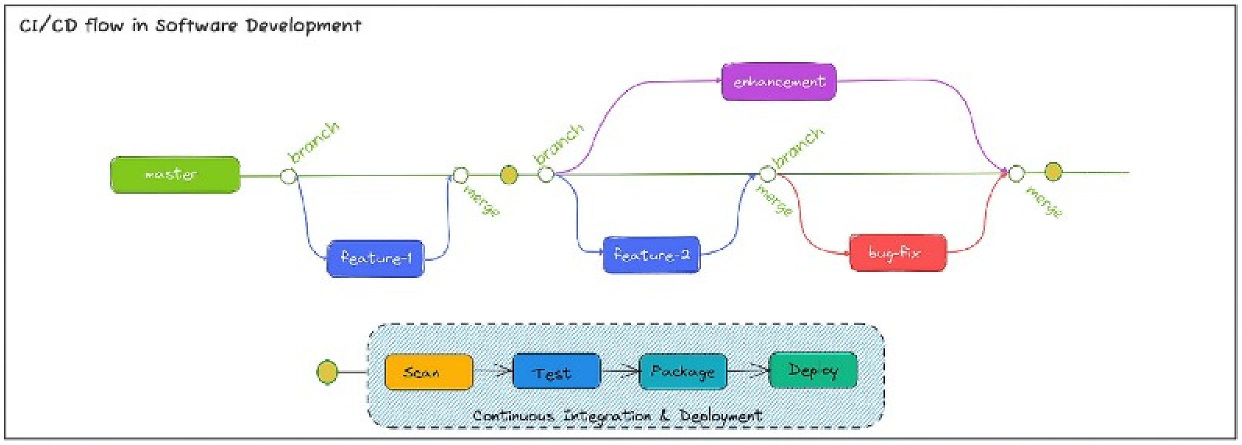
You might have heard of CI/CD (Continuous Integration/Continuous Deployment) where software engineering teams develop an application, seamlessly integrate new changes, bug fixes & enhancements and deploy the latest application. This works in software scenarios as there are tools and technologies available to do this.

Let’s take the example of building a simple web app. A minimal team for building a web app would consist of front-end engineers, back-end engineers, database admins and deployment engineers.



Web-App Dev Team (Image by Author)

The teams would have their code repositories in GitHub or some repository hosting service. As the new code changes and bug fixes are patched, the developers will start pushing the code to the features-and-fix branches of the repository. Once the code is verified, a pull request is raised to merge the changes with the main branch; after merging the latest code a pipeline is triggered to scan and test the code, once passed the code can be packaged and deployed.



CI/CD flow in software development (Image by Author)

Now imagine a machine learning (ML) and data scientists team trying to achieve the same, but with an ML model. There are a few complexities involved -

- Developing ML model isn't the same as developing a software. Most of the code is essentially a black-box, difficult to pinpoint issues in ML code.

- Verifying ML code is an art unto itself, static code checks and code quality checks used in software code aren’t sufficient, we need data checks, sanity checks and bias verification.
- ML model performance depends on the data used to train and test.
- Software Development has standardized architecture and design patterns, ML doesn’t have a widely adopted design patterns, each team may follow their own style.
- Multiple classifiers and algorithms might be used to solve the same problem.

All these complexities raise the important question—**Can ML be performed in a continuous integration style?**

This is where CML comes into the picture. **CML** is an excellent opensource tool by **Iterative.ai**, it allows data scientists and machine learning teams to perform continuous training and integrations of models using the existing sites such as **GitHub, GitLab,** and tools such as **Docker** and **DVC**. No additional tech stack is required.

Now you might say—“But I am quite happy to develop my ML models in my notebook and run it, I can change parameters whenever I want to, tweak the code as I wish. Why would I need CML?”

Why would we need CML?

Consider the process of getting an ML model to production—a bunch of data scientists work on the problem, fetch the data, build a model, train it on the data, test and evaluate it. If it is good enough, they will deploy it. In this process, we need to ensure certain checks and balances are in place -

- Can a model created by one data scientist be reproduced by others on different systems and environments?
- Is the proper version of data being used for training?
- Are the performance metrics of the model verifiable?
- How do we track and ensure the model is improving?
- Most of the teams would use a cloud GPU instance to train the model—this might be an on-demand instance to reduce cost (needed only for the duration of training). How do we make sure the proper environment is set up for training?

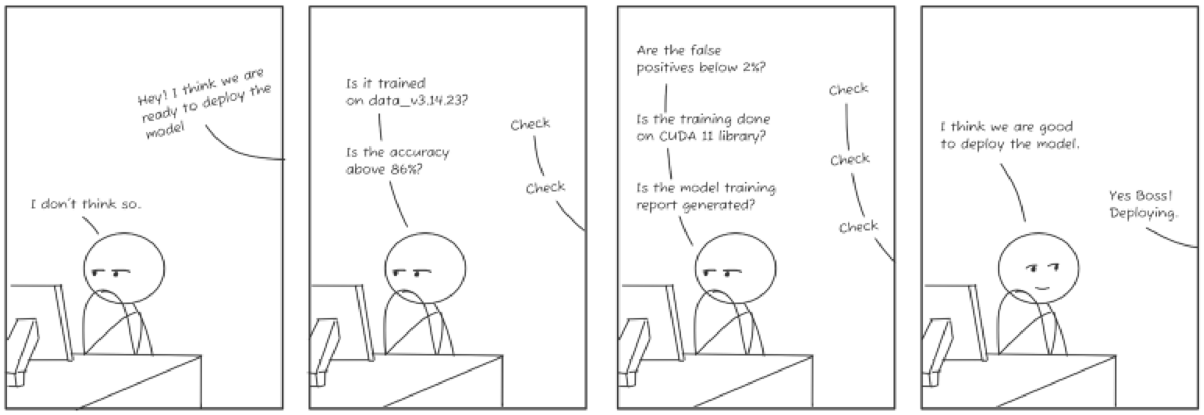
- Once the training is complete, is there a report that gives us the details of the training session for comparison later?
- Would the same checks happen during the retraining of the model?

These steps are necessary to ensure that the model improves over time. This becomes extremely tedious to keep track of as the teams get bigger or as the model becomes more complex.

***CML* improves & streamlines this process; introducing a continuous workflow for provisioning cloud instances, training model on them, collecting metrics, evaluating model performance and publishing summary reports. It allows the developers to introduce a series of checkpoints that are easier to track and reproduce across different scenarios.**

CML workflows make all of this possible with no extra tech stack and with a couple lines of code integrate seamlessly into the existing development process that most of the developers are familiar with.

This adds a layer of visibility to the development process, data scientists now have access to multiple training reports, and metrics to compare and evaluate the model. This helps teams to develop and take the model to production faster, it reduces a lot of manual effort and the rework involved in fixing the model if there is a degradation in the performance.


Hey! I think we are ready to deploy the model
I don't think so.
Is it trained on data_v3.14.23?
Is the accuracy above 86%?
Check
Check
Are the false positives below 2%?
Is the training done on CUDA 11 library?
Is the model training report generated?
Check
Check
Check
I think we are good to deploy the model.
Yes Boss! Deploying.

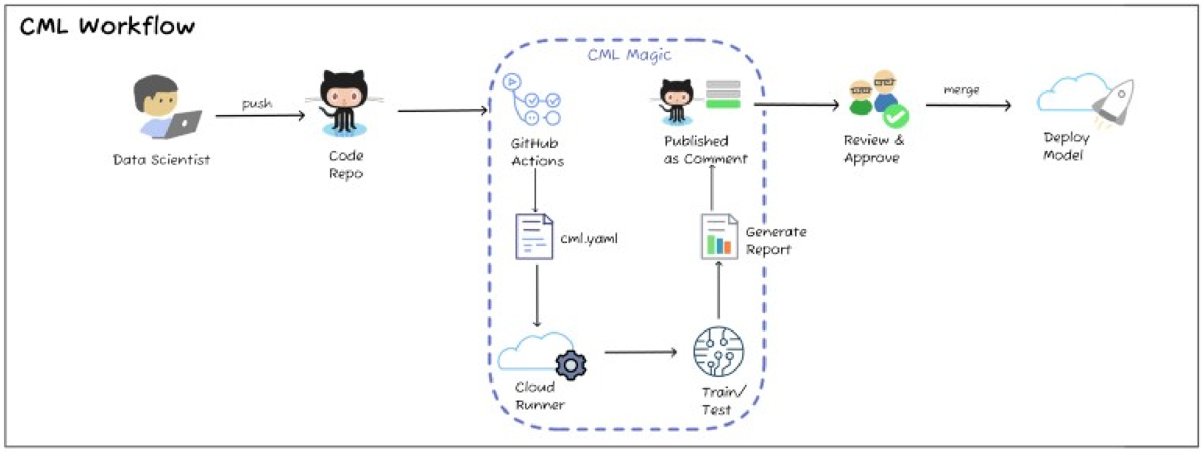
Let us deploy (Image by Author)

Let's see how it works.

How does CML work?

CML injects CI/CD style automation into the workflow. Most of the configurations are defined in a `cml.yaml` config file kept in the repository. In the example below this file specifies what actions are supposed to be performed when the feature branch is ready to be merged with the main branch. When a pull request is raised, the GitHub Actions utilize this workflow and perform activities specified in the config file - like run the `train.py` file or generate an accuracy report.

CML works with a set of functions called **CML Functions**. These are predefined bits of code that help our workflow like allowing these reports to be published as comments or even launching a cloud runner to execute the rest of the workflow. You can see in the below workflow how all these steps are condensed into one step



CML Workflow (Image by Author)

Let's understand this with an example

Example

Suppose 3 data scientists—**Anya, Jack and Nora are working on building a classifier to recognize and classify different fruits.** Anya starts with a basic binary classifier that recognizes a fruit and classifies it as Apple or Not Apple. Then Jack proceeds to add a capability of recognizing Orange and Banana.

All Jack has to do

1. Create his feature branch

2. Write his code, run his training and testing on the previous model
3. Push his code to the feature branch
4. Submit a pull request to merge his code to the main branch

CML jumps in when the pull request is made and runs the actions given in the `cml.yaml` config file. A report is generated, then the team can decide whether to merge Jack's code or not. No duplicate files, no multiple copies of same notebook, just a streamlined workflow.

So what kind of magic exactly happens in this config file? Let's take a look

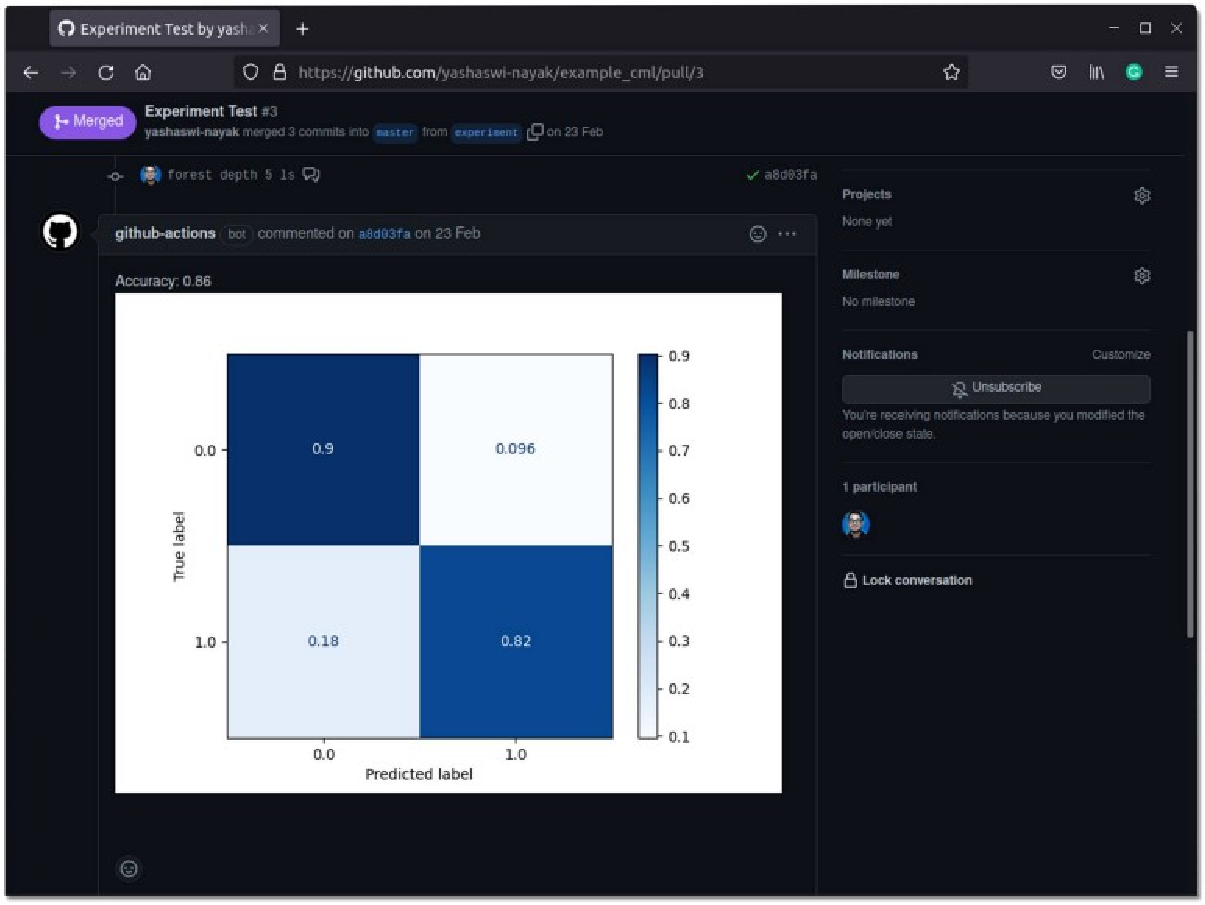
This `cml.yaml` file is a GitHub Actions script. Don't worry if you get confused at first, I will simplify it for you.

This file tells GitHub the following points

- **What is the name of the action workflow**—model-training
- **When should this workflow be performed**—on pull request to main
- **What jobs should it perform**—train-model
- **Where should it perform**—On a GitHub runner with the latest Ubuntu OS (we don't use a cloud GPU runner here)
- **What steps should it perform**—Train model
- **What does it need to perform this step**—checkout, setup-python, setup-cml. These are actions available in GitHub and its marketplace. Actions are bits of code somebody has already written to perform a task, which you can easily reuse in your script. So the latest code of yours will be checked out on the temporary Ubuntu virtual machine, python and cml will be installed on that machine.
- **What environment variables should be used**—The REPO_TOKEN is given by GitHub, so the virtual machine knows only authorized GitHub repo is allowed to perform this task.
- **What commands should be performed as part of this step**—Install the python packages, run the *train.py* file, publish the evaluation reports to a file called `report.md`. This will show the report data like accuracy, evaluation metrics and error metrics, whichever you wish to specify as a part of your model evaluation process.

- **CML-Functions**—You notice the `cml publish` and `cml send-comment` commands in the last two lines of the script, these are **CML Functions**. These functions simplify your task of creating a detailed report of model training in the pull request. You can create a cool visual report outlining the performance of the model. This helps the data scientist to figure out which pull request can be approved for the final merge.

In a nutshell, all Jack did was write his code, test his model, push his changes to his branch and then raise a pull request. The rest of the steps of model evaluation, comparison and result tabulation was taken care of by CML. After the CML actions are complete a cool training session report is available for other data scientists to verify and evaluate—in the same repo, as you can see below.


Experiment Test by yash
https://github.com/yashaswi-nayak/example_cml/pull/3
Merged
Experiment Test #3
yashaswi-nayak merged 3 commits into master from experiment on 23 Feb
forest depth 5 1s
a8d03fa
github-actions bot commented on a8d03fa on 23 Feb
Projects
None yet
Milestone
No milestone
Notifications
Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.
1 participant
Lock conversation
True label
Predicted label

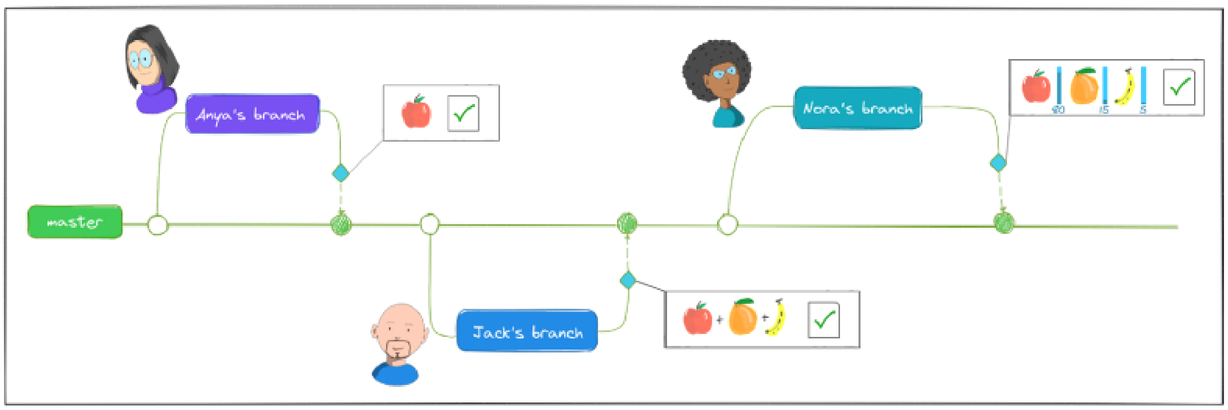
Accuracy: 0.86

	0.0	1.0
0.0	0.9	0.096
1.0	0.18	0.82

CML Report (Image by Author)

Let's say Nora wants to use the latest code and train with her data, wants to add the capability of giving percentage-wise classification. So her model classifies the image of fruit as 96% Apple or 60% Banana, she has a large data set of images to be trained on and a GPU would save her a lot of effort. ***But Nora doesn't know all the system setup and configurations required for setting up a CUDA GPU based instance.***

Here the CML comes to the rescue as well. She could use a custom cloud runner and connect to an AWS GPU instance and train her model without having to learn additional tech or system setup knowledge. All she has to do is configure the `cml.yaml` file to use the AWS and GPU settings. The workflow would then use a custom GPU based container to train the model and publish her result.



CML Magic (Image by Author)

Now every time someone creates a feature, model enhancement, performance improvement or a bug fix, the evaluation report will be readily available during the merge. Multiple data scientists could be working on parallel versions of a model, say different classifier algorithms at the same time, using CML they can see if the models are trained properly, deployable and compare metrics; all in the same repository workflow.

Excellent! What next?

If you are a data scientist or machine learning team, first create your repository on GitHub, GitLab or Bitbucket. Proceed to https://github.com/iterative/cml follow the steps given there to kick off your journey into CML.

You can try out the following use cases

- View model accuracy and performance metrics report of your model training

- Use **CML** to run your training on different versions of data using **DVC**, pull new data, train your model, publish your report and push your latest model to your storage.
- If you have a **TensorFlow** model, you can connect to tensorboard.dev and publish your report regularly using the CML workflow
- Use a custom CML runner for your project and train your model on your **AWS EC2, Azure, GCP** cloud instances, or your on-premise machines
- Use an AWS GPU based instance without extra setup, the instance can be removed as soon as the training is done. Helps to reduce training and instance costs.
- Since the workflow is using GitHub Actions at the base level, you can extend the scripts to other workflow actions as well—such as creating a model pickle file and storing it in the remote storage on every push to a specific branch.
- Add CML on top of GitHub Actions workflow, it's easy to integrate it with existing workflow—say you wish to keep track of major version of data and model change, you can use CML to train the model, generate the report during pull request of major versions and add a simple GitHub Action to mail that report.—The CML + Github Actions possibilities are ♾️ .

That's all for now. Happy CML adventures 🙂

Links and References

- **CML**—https://cml.dev/
- **Github Repo**—https://github.com/iterative/cml
- **CML Youtube Series**—https://www.youtube.com/playlist?list=PL7WG7YrwYcnDBDuCkFbcyjnZQrdskFsBz