QUANTORI

QFlow: Solution for Managing ML Projects

Alexander Knop
Principal Mathematician, Quantori

8 January 2024

White Paper

TABLE OF CONTENTS

EXECUTIVE SUMMARY

In recent years, the rapid growth of AI technologies has resulted in an increasing number of organizations adopting machine learning techniques to solve complex business problems. However, the process of building and deploying ML models at scale can be challenging, involving numerous complex steps such as data preparation, feature engineering, model training, and deployment.

Various ML frameworks have emerged to address these challenges. This document will present a framework that provides a structured approach to managing ML projects, enabling teams to collaborate more efficiently, improve code quality, and automate various tasks. Like popular frameworks such as Kedro, Flyte, and SageMaker, this framework provides an opinionated structure to develop, maintain, and scale ML projects. The framework emphasizes modularity, reproducibility, and versioning, enabling teams to track changes easily and reproduce results.

In the following sections, we will dive into this framework's key features and benefits, along with examples and best practices for implementing it in your ML projects.

CAUTIONARY TALES

As with any other software projects, ML and data science projects require adherence to best practices, failure to follow these practices can lead to various issues, including project delays, technical debt, and poor model performance.

This section will present archetypical examples of machine learning projects that have gone awry due to engineers not following best practices. We will explore the specific challenges these projects faced, the root causes of their failures, and how they could have been avoided using proper ML project management frameworks.

The Tale of Broken Notebooks

In this project, a team of data scientists analyzed datasets consisting of gut microbiome data to investigate the effects of certain drugs improving gut microbial composition on treating other diseases. The team successfully developed and trained machine learning models to predict the impact of gut microbial composition on the treatment. They also used various visualization techniques to present the findings to their client, who was pleased with the results.

However, problems arose when the client attempted to reproduce the experiments independently. Despite the successful delivery of the results, the client was unable to reproduce the findings due to several issues with the project's development and management practices. Specifically, the team had not fixed the versions of the libraries used in the project, which led to discrepancies between the client's environment and the team's.

Furthermore, the team developed the project using Jupyter notebooks, which are difficult to test and reproduce; as a result, changes in a shared library made some of the notebooks unrunnable, exacerbating the client's difficulties in reproducing the results.

One way to ensure code reproducibility and version control in ML projects is using scripts instead of notebooks. Scripting allows for proper testing of the code using standard libraries and frameworks. In addition, thorough project documentation, including dependencies and their specific versions, is crucial for enabling future reproducibility.

The Tale of Growing Pains

In this project, a client approached us to help scale their analysis of health-related surveys and imaging data. Upon reviewing the code, the team discovered it was developed using Jupyter notebooks. This approach resulted in code that was repetitive, contained bugs that had been copied over and over, and was challenging to parallelize, which meant that the team could not train models on larger datasets. Additionally, the team had yet to implement hyperparameter tuning processes beyond a simple brute-force search, limiting the models' accuracy and robustness.

Finally, the lack of configuration in the codebase incentivized developers to hardcode the dataset's location and heavily rely on its structure. This approach made it challenging to scale the project since changes in the dataset's structure would require significant code modifications.

They could have avoided this situation if the team had used a framework that offered cloud-based development and deployment capabilities. With such a framework, the team could have used the cloud infrastructure's scalability and parallelization capabilities to train models on larger datasets and tune hyperparameters more efficiently.

The Tale of Missing User Interface

In this project, our team was tasked with developing a solution for automatically labeling microscopic images. Initially, the client's requirements stated that they would need to analyze at most a couple of thousand images daily, and a console application would suffice. However, after the initial solution was delivered, the client requested that we analyze a dataset of hundreds of millions of images and create a dashboard or user interface to run the inference since the console was not user-friendly for the scientists utilizing the solution. In addition, the client wanted to run the inference in the cloud since their on-prem server was unavailable.

The change in requirements presented significant challenges for our team. We had to re-architect the solution to make it scalable and deployable in the cloud and create a dashboard or UI that scientists could use to run the inference. This was a time-consuming and complex process, and it delayed the project significantly.

However, this story would have been much more straightforward if the team had used a framework that allowed us to scale and create dashboards automatically. With such a framework, we could have easily deployed the solution to the cloud and created a dashboard or UI for scientists without needing significant additional development effort. This would have enabled us to deliver the solution to the client faster and more efficiently, saving time and reducing costs.

In summary, when developing ML projects, it's essential to consider scalability and deployment requirements from the outset, even if the initial requirements seem small. By using a framework that allows for easy scaling and dashboard creation, teams can simplify their development process and respond more quickly to changing client needs.

ALTERNATIVES

As mentioned, when it comes to managing machine learning projects, various tools and frameworks are available in the market, each with its own strengths and weaknesses. In this section, we will compare QFlow, a new pipeline management solution, with some of the existing tools in the market, including Kedro, Flyte, and SageMaker. While all these tools have their own merits, we will argue that QFlow is ultimately a better solution, offering a more flexible, scalable, and intuitive approach to managing ML pipelines.

Kedro

Compared to Kedro, QFlow offers a different approach to writing ML pipelines. Rather than explicitly describing computation graphs, QFlow allows for describing workflows using a Python-based domain-specific language (DSL) that is more readable and easier to understand for non-technical stakeholders. This makes it easier for the team to collaborate and update the pipeline when necessary.

Another critical difference between QFlow and Kedro is the ability to run workflows in the cloud. With QFlow, the team can provision cloud infrastructure for running workflows, making scaling and running large-scale pipelines easier. This also enables the team to take advantage of the scalability and flexibility offered by cloud infrastructure, making it easier to manage complex pipelines that require significant computing resources.

In summary, QFlow offers a more flexible and scalable approach to managing ML pipelines, allowing for the description of workflows using a Python-based DSL, running workflows in the cloud, and provisioning infrastructure as necessary. These features make it easier for teams to manage complex ML projects, collaborate more effectively, and scale their pipelines as necessary.

Flyte

QFlow and Flyte are pipeline management solutions using a DSL to describe pipelines. However, QFlow and Flyte differ in several key areas.

One significant difference between QFlow and Flyte is their approach to the project structure. QFlow enforces a structured project layout, making organizing and maintaining a complex ML project easier. In contrast, Flyte does not impose any particular project structure, allowing for more flexibility in how pipelines are organized.

Another difference is in their support for tracking. QFlow provides extensive tracking functionality, allowing users to track experiments, artifacts, and metadata associated with a pipeline run. On the other hand, Flyte does not have built-in tracking support, making it more challenging to monitor pipeline runs and compare results across different runs.

A significant advantage of Flyte is its support for writing pipeline steps in different programming languages, whereas QFlow currently only supports Python. However, to use Flyte, artifacts need to be Protobuffer convertible, which can limit the flexibility in how users can represent their data.

While both QFlow and Flyte offer DSL-based pipeline management solutions, their focus and strengths differ. QFlow provides a more structured and intuitive approach to managing ML pipelines, with built-in tracking and scalability features. At the same time, Flyte offers more flexibility in writing pipeline steps in different programming languages. However, this flexibility can present challenges for many ML projects, particularly those with limited engineering expertise.

SageMaker

Amazon SageMaker is designed to run pipelines on AWS and provides various built-in components for data processing, model training, and deployment. However, SageMaker has some limitations regarding portability. The solution requires projects to be locked into the AWS environment and has limited support for running code on-premises. Furthermore, the explicit graph description and separate scripts for each step can complicate testing and support.

QFlow, on the other hand, provides a more structured and portable approach to pipeline management. As mentioned, using a python-based DSL, QFlow, allows for a more readable and intuitive way of describing workflows. Additionally, QFlow offers the ability to run workflows in the cloud, provision infrastructure, and track experiments out of the box. This makes QFlow a more comprehensive and efficient solution for managing ML pipelines, especially for teams that may not have extensive engineering expertise or that need to work across multiple environments.

LIFE SCIENCE

QFlow provides numerous benefits for life science projects, particularly for startups looking to scale. First, the easy-to-use python-based DSL allows for straightforward pipeline management without the need for extensive engineering expertise. This is particularly important for life science projects, where scientists and researchers may not have a strong background in software engineering.

Additionally, QFlow enables the deployment of specialized tools often used to process life science data. This allows for greater flexibility and customization in the pipeline management process, essential for life science projects that often require unique and complex workflows.

Finally, QFlow's scalability features make it an ideal solution for startups in the life science industry. As the volume of data grows, QFlow allows for easy scaling to meet the project's demands without the need for significant infrastructure changes. This scalability also enables startups to more easily handle larger datasets and accelerate the research process, which is crucial in the competitive world of life science.

TYPICAL WORKFLOW WHEN USING QFlow

A typical workflow of using QFlow starts with creating a project and configuring it according to the specific needs of the ML pipeline. This can be done using a simple configuration wizard that guides the user through the process and helps set up the project's environment, dependencies, and resource allocation.

```
(qflow-py3.10) ➜  ml-pipelines git:(dev) ✗ qflow new --project-name example
Tracker to configure (mlflow, wandb, filetracker) [filetracker]: mlflow
SUCCESS: Project 'example' is created successfully.
Configuring the component: 'mlflow':
Name of the new configuration [mlflow]:
Address of local or remote tracking server [.qflow/tracker/mlflow]:
Address of local or remote model registry server [null]:
SUCCESS: Component mlflow configured
SUCCESS: Project 'example' is configured.
```

Fig. 1. Creating a New Project

Once the project is set up, the user can prototype different pipeline steps using Jupyter notebooks or scripts. As soon as they are ready, these notebooks/scripts can be converted into pipeline steps; the pipeline is written using QFlow's DSL-based approach, which allows for intuitive and readable pipeline definition. The pipeline definition and the steps are version-controlled using GitHub. They can be executed from the command line using QFlow's CLI tools or executed from the PR automatically to allow the review of the metrics and visualizations.

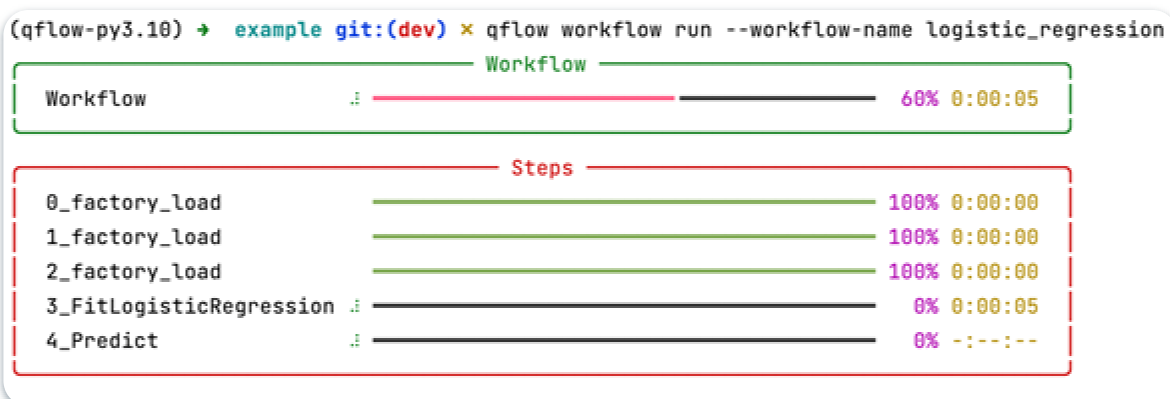

Fig. 2. Creating Pipelines from CLI

With QFlow, it is also possible to deploy specialized tools and libraries needed for the data processing phase (e.g., it allows deploying a service for fMRI processing), which is often a critical part of life science projects. This is achieved by leveraging containerization and the support of plugins.

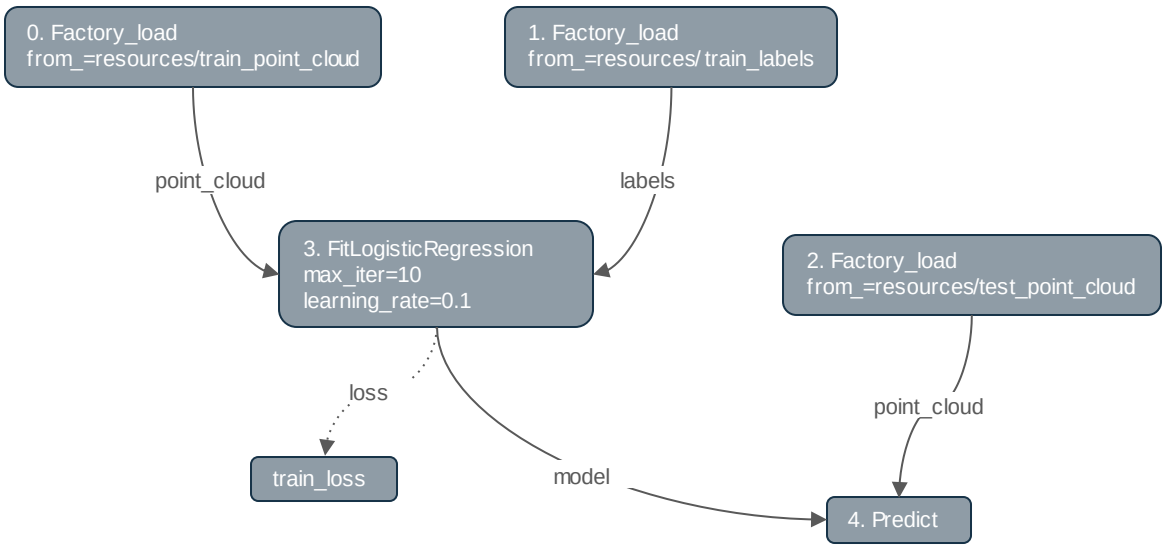

Fig 3. Example of a Pipeline Graph

As the project grows in size and complexity, the QFlow project can be configured to run the pipelines in the cloud to handle larger datasets and more complex models.

Overall, QFlow's intuitive interface, ability to configure the project to specific needs, and scalability make it an excellent choice for life science projects, where data processing and scaling are essential.

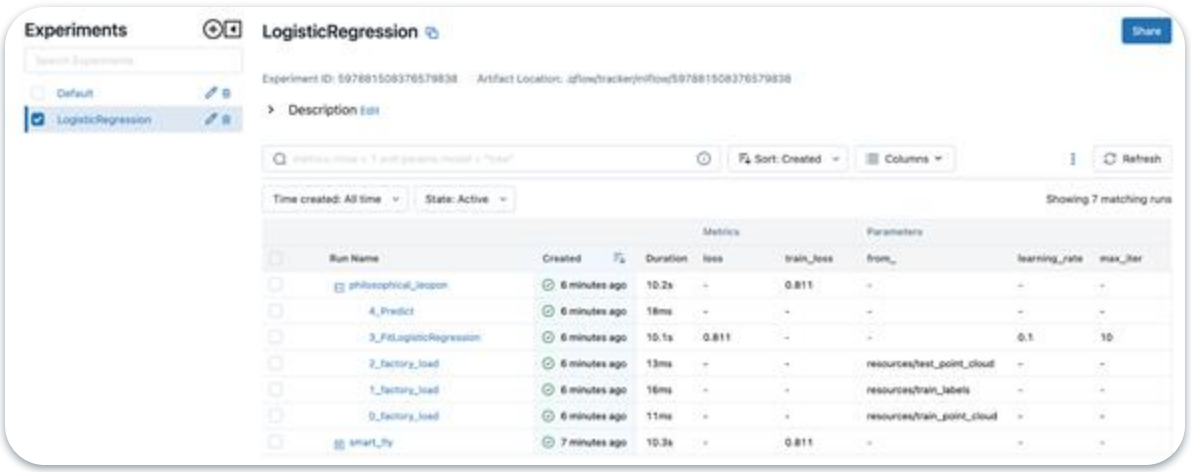

Fig 4. Tracked Pipeline

To expand QFlow's functionality, it uses pluggy, a lightweight plugin manager for Python. This feature allows developers to create custom plugins and add new functionality to QFlow as needed.

QFlow integrates with popular machine learning tracking solutions such as MLFlow and Weights & Biases for tracking purposes. Custom tracking solutions can also be added by creating plugins.

For infrastructure provisioning, QFlow uses Terraform, a widely used infrastructure as a code tool. This enables developers to provision new resources and infrastructure quickly and easily in a scalable and reproducible way without requiring in-depth knowledge of the cloud.

Finally, QFlow provides automation for GitHub workflows, enabling users to run and test pipelines from pull requests. Model registries like SageMaker and MLFlow registry are also integrated into QFlow for deploying models.

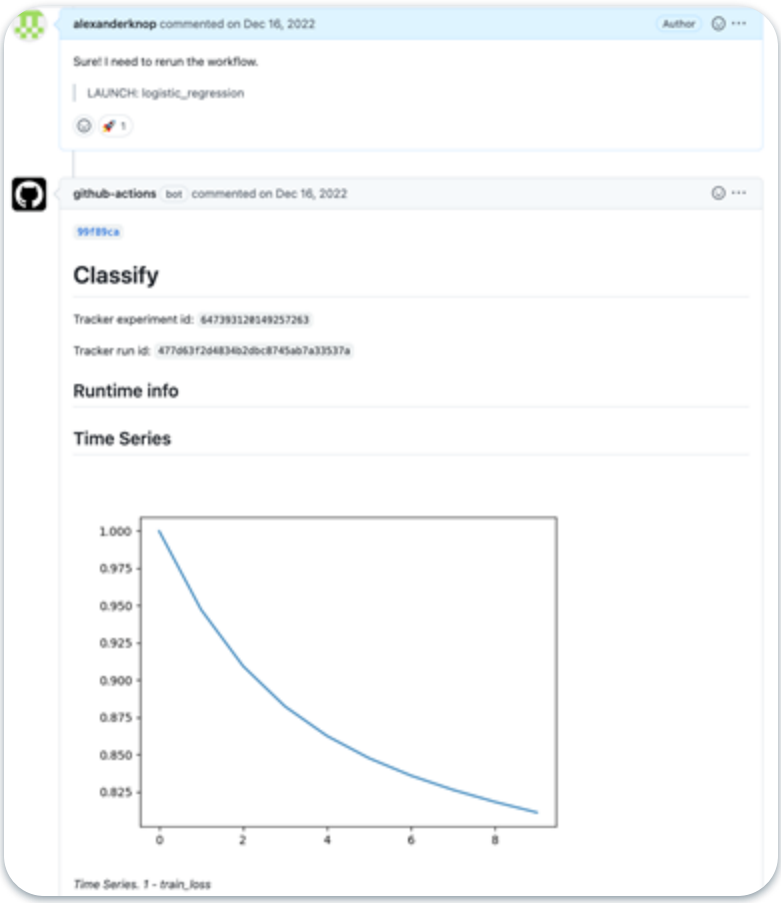

Fig 6. Pipelines can be Executed from PR

TECHNICAL DETAILS

QFlow is a pipeline management solution that builds upon a range of open-source libraries and tools. At its core, QFlow leverages Python to define and execute pipelines. Python's metaprogramming capabilities are used to convert the pipeline DSL into a computational graph. The user-friendly CLI is provided by typer, a library for building command-line interfaces. Hydra, a configuration management framework, is used to create hierarchical configurations that can be overridden at runtime.

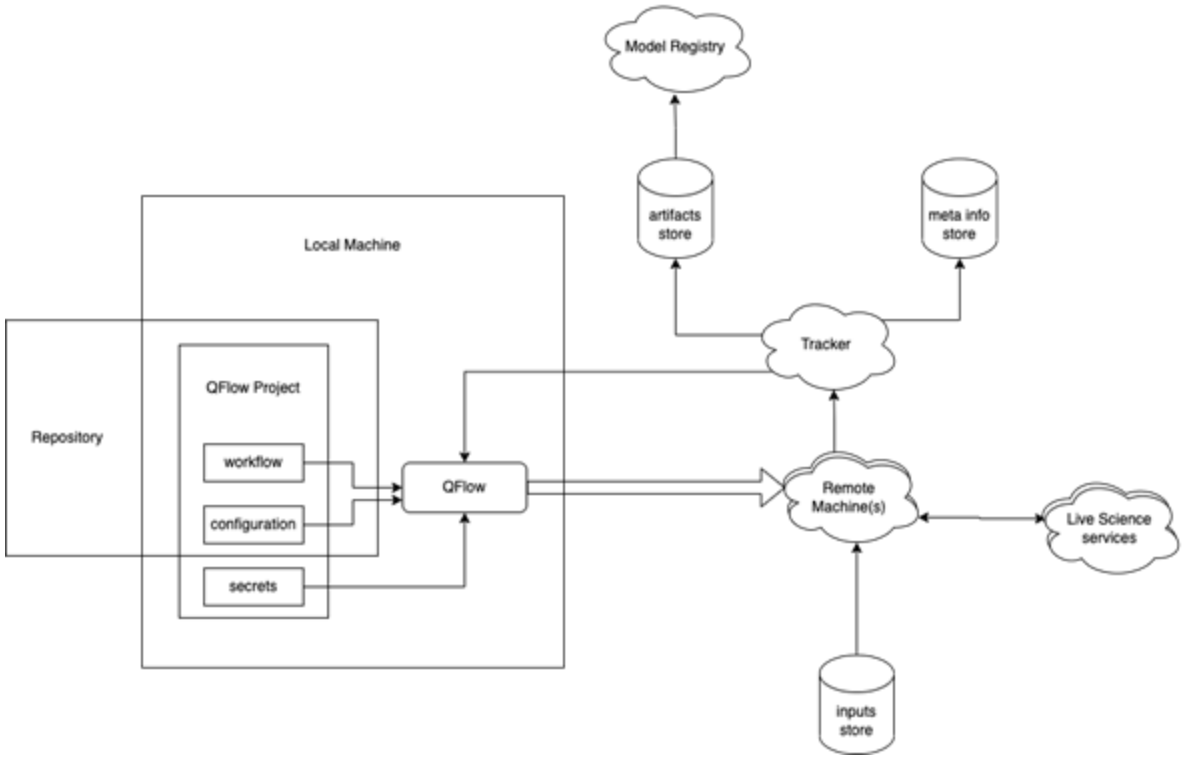

Fig 5. Diagram of a Running Pipeline

We develop cutting-edge technology systems, applications, and infrastructures for biotech, pharmaceutical, and healthcare companies that accelerate drug discovery and improve patient outcomes. Our innovative approach harnesses the power of data engineering and informatics, machine learning, emerging technologies, and cloud expertise to advance research and development and ultimately bridge the gap between meaningful data and patient success.

Alexander Knop

Principal Mathematician, Quantori

8 January 2024