

# Data Version Control (DVC) in Machine Learning Projects

Nhóm MLOps

Ngày 18 tháng 10 năm 2025

Nội dung về Data Version Control (DVC) sẽ chia thành 5 phần chính:

- **Phần 1: Tổng quan về AI, MLOps và Data Versioning**
- **Phần 2: Thách thức trong Quản lý Dữ liệu và Code**
- **Phần 3: Giới thiệu về Data Version Control (DVC)**
- **Phần 4: Case Study: Triển khai DVC cho Dataset MNIST**
- **Phần 5: Tổng hợp Pipelines và Cờ khởi nimm Versioning**

## Phần 1: Tổng quan về AI, MLOps và Phiên bản hóa Dữ liệu (Data Versioning)

### 1.1 Data Versioning là gì ?

**Data Versioning** (phiên bản hóa dữ liệu) là một hệ thống giúp quản lý và theo dõi những thay đổi của dữ liệu và mô hình trong suốt vòng đời của Machine Learning (ML).

Tất nhiên không chỉ có phiên bản khác nhau của bộ dữ liệu (dataset), mà còn quản lý các file cấu hình (Config), tham số (parameters) và kết quả đánh giá (Eval result). Về bản chất, nó hoạt động như "Git cho dữ liệu", giúp kiểm soát phiên bản code và theo dõi phiên bản dữ liệu nhằm xác minh code và dữ liệu huấn luyện và tạo ra mô hình.

### 1.2 Vì sao cần Data Versioning?

#### 1.2.1 Góc nhìn nghiệp vụ (Business Perspective)

Từ góc nhìn nghiệp vụ, AI có thể xem là một chức năng (function) giải quyết các bài toán cụ thể, ví dụ như phân tích cảm xúc (Sentiment Analysis) hay đề xuất hệ thống gợi ý (Recommendation) trong một hệ thống phân tích dữ liệu. Vì vậy, việc quản lý phiên bản code và dữ liệu là rất quan trọng để đảm bảo tính nhất quán và khả năng tái lập của mô hình.

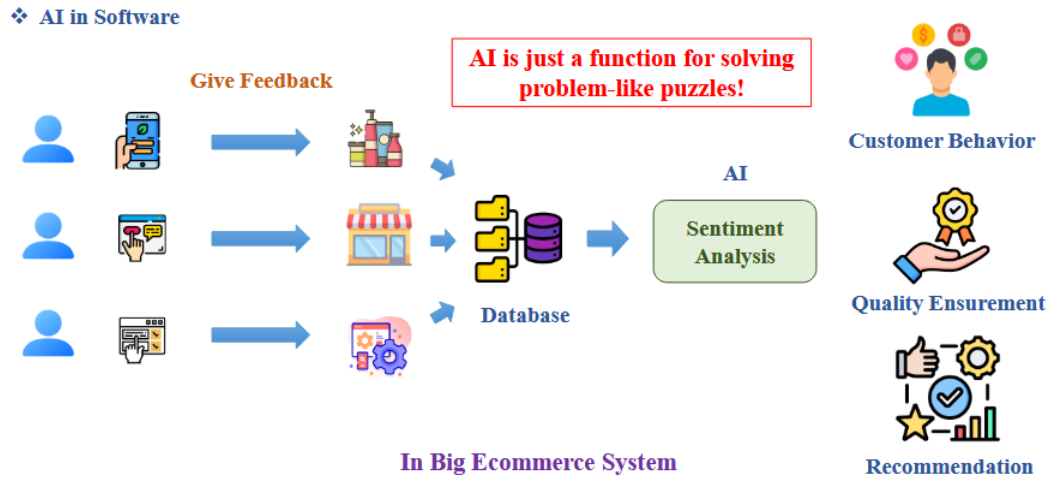


Figure 1: Vai trò của AI trong một hệ thống Thương mại điện tử

Toàn bộ quy trình ML bắt đầu từ **Yêu cầu Nghiệp vụ (Business Require)** và đi qua một vòng đời (life cycle) hoàn chỉnh. Một bộ chỉ số (Quality Ensurement) và giám sát (Monitoring) mô hình AI một cách hiệu quả, doanh nghiệp phải chú ý khi triển khai các cấu hình:

- Mô hình đang chạy trong production có huấn luyện dữ liệu mới?
- Khi mô hình đưa ra kết quả sai, làm thế nào để tái lập (reproduce) lại?
- Nếu dữ liệu mới làm giảm hiệu suất, làm sao để quay lại phiên bản mô hình trước?

Data versioning cung cấp khả năng **truy xuất nguồn gốc (lineage)** này, cho phép theo dõi dòng chảy của code, dữ liệu, tham số để tạo ra mô hình mới.

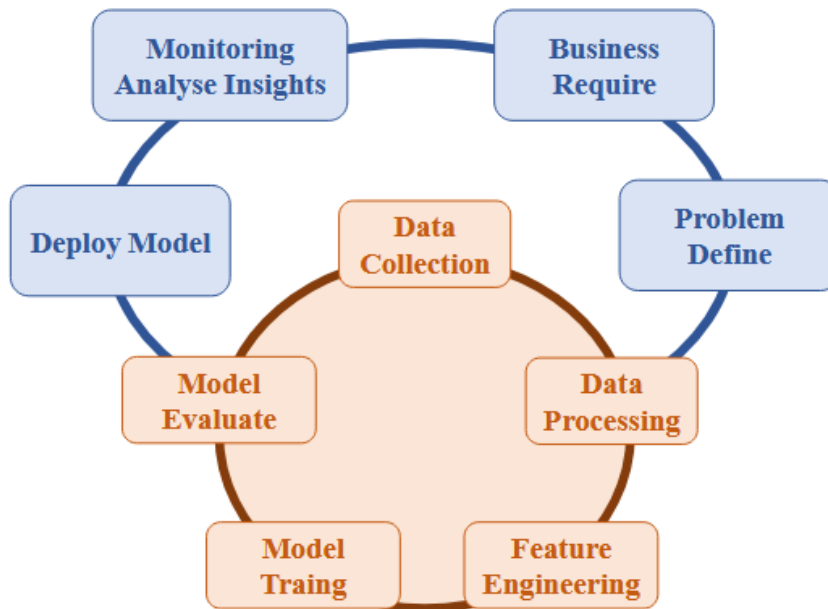


Figure 2: Vòng đời Machine Learning (ML Life Cycle)

### 1.2.2 Góc nhìn MLOps

MLOps tập trung vào việc tự động hóa quy trình chuyển giao mô hình từ môi trường nghiên cứu (Research Environment) sang môi trường triển khai (AI Service). Quy trình này bao gồm các bước lập kế hoạch xử lý dữ liệu (Data Handling), Huấn luyện mô hình (Model Training), và đánh giá mô hình (Model Evaluation).

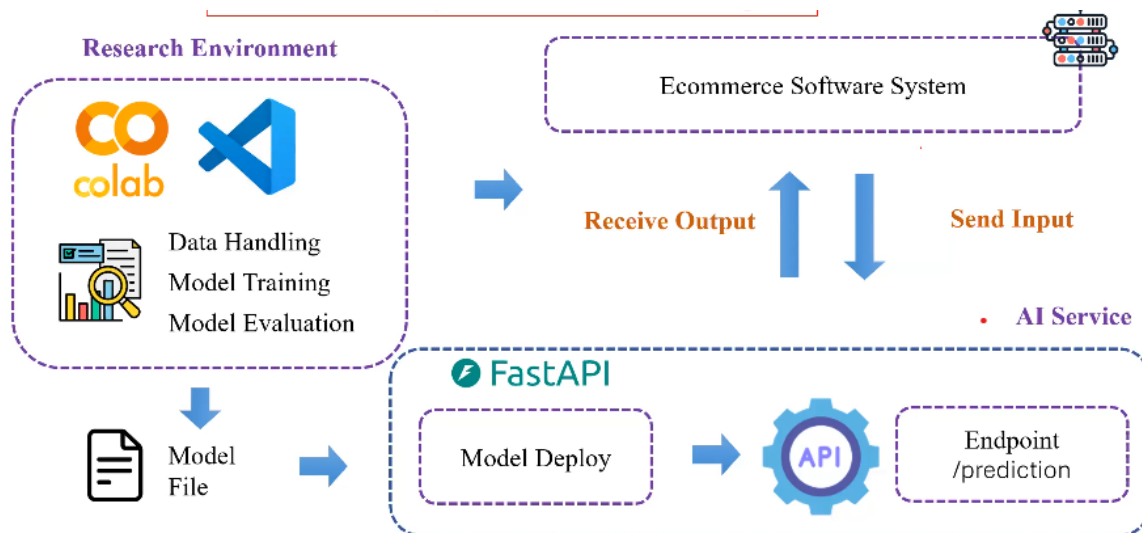


Figure 3: Quy trình MLOps cơ bản

Data versioning là thành phần quan trọng để quản lý các phiên bản dữ liệu. Nó giúp tự động hóa việc quản lý các "vật thể" (artifacts) của mô hình (như dữ liệu đầu vào, file mô hình, file kết quả) và bảo tồn lịch sử của các phiên bản, đặc biệt là khi làm việc nhóm.

## Phần 2: Thách thức trong Quản lý Dữ liệu và Code

Khi thực nghiệm trên tập dữ liệu lớn, các vấn đề như các thành phần trong nhóm "mô hình / ngân hàng" về dữ liệu, code, tham số phiên bản mô hình liên quan đến dữ liệu trong một nhóm. Một lợi ích của dữ liệu dạng TimeSeries và liên quan và phi đồng nhất liên tục. Hơn nữa, trong quá trình này, mô hình sẽ đi qua 2 vòng đời khi CÚ và KHÔNG CÚ data versioning pipeline.

### 2.1 Vòng đời thực tế: Th nghiệm mô hình Time Series

#### 2.1.1 Khi KHÔNG có Data Versioning (Cách làm thủ công)

Giai đoạn đầu tiên của quá trình phát triển mô hình dữ liệu chuỗi thời gian (time-series) phức tạp. Quy trình này tạo ra các file: 'dataset files' (dữ liệu gốc), 'parameters files' (tham số), 'checkpoint files' (kiểm tra mô hình), và 'dataset augment files' (dữ liệu tăng cường).

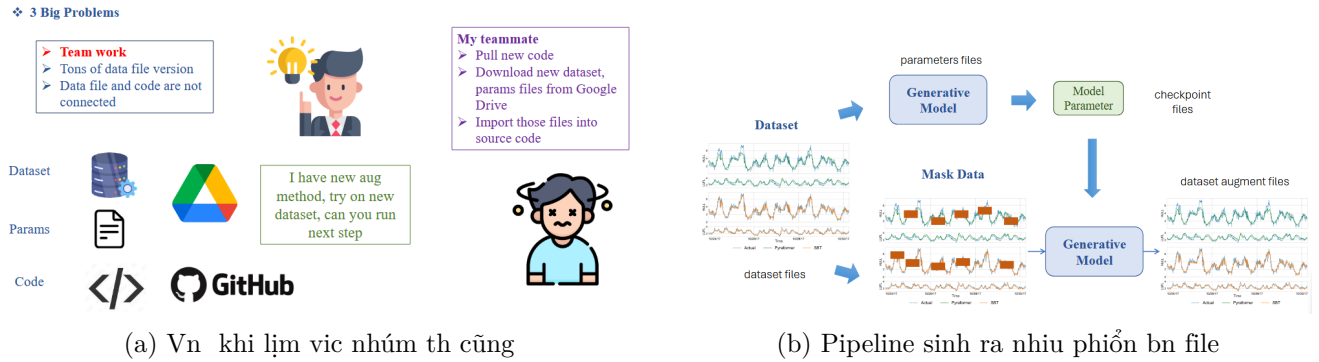


Figure 4: Sự hỗn loạn khi quản lý thử nghiệm các file dữ liệu vị thí nghiệm

Khi một thành viên trong nhóm ("My teammate") muốn chạy thử nghiệm mới, quy trình cả hai thì cũng vị làm chậm quá trình phát triển:

1. **Pull new code:** Tải code mới nhất từ GitHub.
2. **Download new dataset/params:** Với Github không cho lưu file lớn hơn 100mb, phải truy cập Google Drive, tìm kiếm file dữ liệu, file tham số mới nhất vị thí nghiệm.
3. **Import files:** Sao chép các file này vị ứng dụng trong source code.

Hiện nay đã có 3 vấn đề: (1) Có quá nhiều phiên bản file dữ liệu, tham số các mô hình khác nhau. (2) Code vị dữ liệu các mới nhất không có sẵn, vị (3) Gây cản trở khi làm việc nhóm. Mới nghĩ sẽ bí rì thì hi "file cnn\_ettm1\_mix1\_mask025.pkl này có hun luy n t dữ liệu, mô hình, thì im nọ ?".

→ Có 1 quy trình (pipeline) thống nhất kiểm soát các phiên bản mô hình, dữ liệu vị code của team có thể hoạt động 1 cách riêng biệt không phụ thuộc vào nhau, ví dụ Google Meet trao đổi nhiều chủ đề nhận được vấn đề thì có thể lập lịch họp riêng.

### 2.1.2 Khi Cần Data Versioning (ví dụ DVC)

Khi sử dụng một công cụ như DVC, quy trình trở nên gọn gàng hơn.

1. **git pull:** Thành viên nhóm tải code mới. Hiện nay, code mới bao gồm các file `.dvc` (siêu dữ liệu) như như trên dữ liệu.
2. **dvc pull:** Hệ thống chỉ cần lấy file `.dvc`, từ đó tìm vị thí nghiệm phiên bản dữ liệu, tham số, vị mô hình tăng vị phiên bản code để tải về (ví dụ: S3, Google Drive, hoặc SSH).

Bằng cách này, DVC giải quyết vấn đề "disconnected" bằng cách liên kết code vị dữ liệu một cách chặt chẽ thông qua các `commit`. Mới thí nghiệm này ghi lại, nhất quán vị có thể **tái lập (reproducible)** một cách chính xác bất cứ lúc nào.

## Phần 3: Giới thiệu về Data Version Control (DVC)

DVC (Data Version Control) là một công cụ mã nguồn mở thiết kế để quản lý dữ liệu vị các dự án Machine Learning, hoạt động song song hỗ trợ cho Git [Ite25].

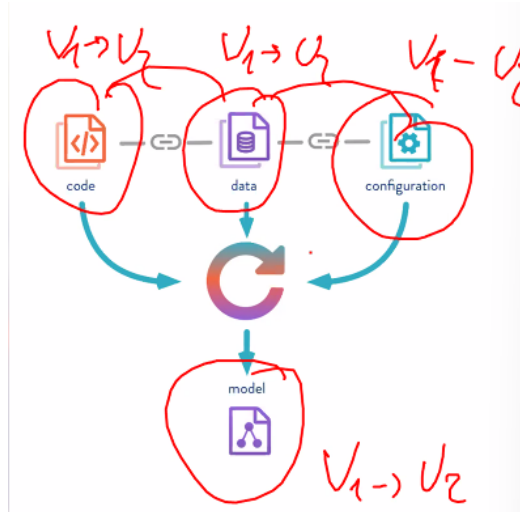


Figure 5: DvC ng b b hóa code, d liu, cu hnh v i mĩ hnh cho m i phiĩn bn, m i khi cú s thay i sau Hun Luyn

### 3.1 Workflow Qun lý Phiĩn bn D liu

Mt workflow phiĩn bn hóa d liu (Data Versioning) l i mt vùng lp qun lý liĩn tc. Quy trnh n i b t u t **Model Training** (Hun luyn Mĩ hnh), s dng mt b **Data Config** (cu hnh d liu) v i **Model (params)** (tham s mĩ hnh) c th. Toĩn b t i sn n i (d liu, tham s, kt qu ònh giò) c h thng **Data & Model Management** theo d i v i lu tr.

Khi cú d liu m i (**New Data**) hoc thay i cu hnh (**Diff Config**), quy trnh **Continuos Training** (Hun luyn Liĩn tc) s c k i ch hot to ra mĩ hnh m i (**New Model**) v i kt qu ònh giò m i (**New Eval result**). Phiĩn bn m i n i l i tip tc c h thng qun lý, hoc sn s i ng cho vic **Model Deployment** (Trin khai).

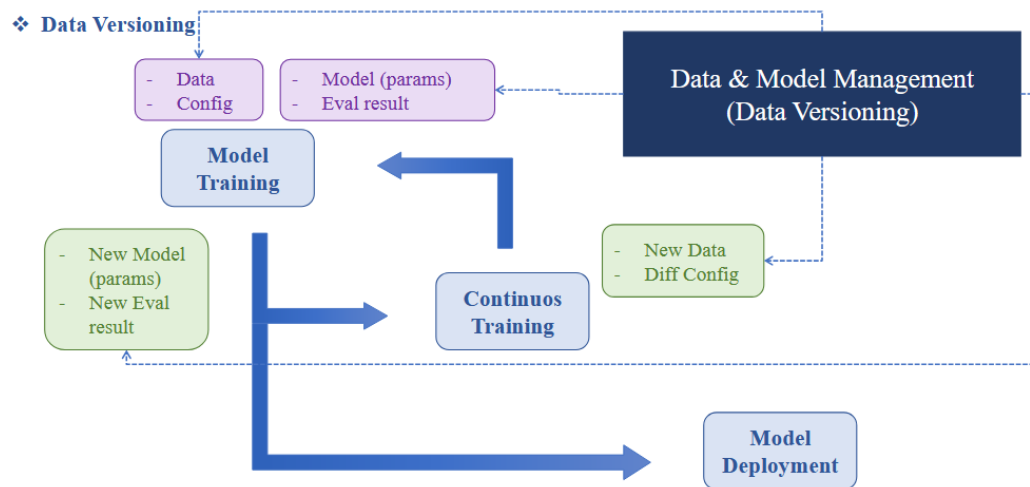


Figure 6: Workflow qun lý d liu v i mĩ hnh trong MLOps

### 3.2 So s i nh DVC v i Gi i thi u chi tit

Trĩn th trng cú nhiu c i ng c phiĩn bn hóa d liu nh Delta Lake, Pachyderm, v i DVC. Tuy nhiĩn, DVC tr nĩn rt ph bin v i c i lý do chĩnh:

- **Mô nguyên m:** S dng gij phõp Apache 2.0.
- **c lp vi nh dng:** **Data Format Agnostic**, ngha lị nú cú th qun lý bt k loi file nịo (model, .csv, .parquet, hnh nh...).
- **c lp vi lu tr:** **Cloud/Storage Agnostic**, h tr hu ht cõc nn tng lu tr ph bin nh S3, GCP, Azure, SSH.
- **D s dng:** **Simple to Use**, vớ cú cõc lnh tng t Git.







	Open Source	Data Format Agnostic	Cloud/Storage Agnostic* (Supports most common cloud and storage types)	Simple to Use	Easy Support for BIG Data
	✓ (Apache 2.0)	✓	✓	✓	✗
	✓ (Apache 2.0)	✗	✓	✗	✓
	✓ (MIT)	✓	✗	✓	✗
	✓ (Non-standard license)	✓	✓	✗	✓
	✓ (Apache 2.0)	✗	✗	✗	✗
	✓ (Apache 2.0)	✓	✗	✗	✓



Figure 7: So sõnh cõc cõng c Data Version Control ph bin

V c bn, DVC lị mt nn tng khoa hc d liu cho phõp bn liõn kt **code**, **data** (d liu), vị **configuration** (cũ hnh) to ra cõc **model** (mũ hnh) cú th tòi lp (reproducible).

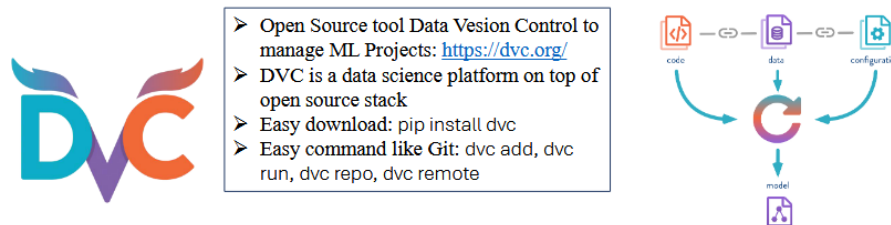


Figure 8: DVC liõn kt Code, Data, vị Configuration qun lý Model

### 3.2.1 DVC khõc Git nh th nịo?

im khõc bit ct lũi lị: **Git qun lý Code** (mõ ngun), trong khi **DVC qun lý Data** (d liu).

Git khõng c thit k x lý cõc file ln (võ d: mũ hnh 500MB). Khi bn dũng DVC, quy trnh lịm vic (Local/Remote) s c tõch bit rũ rịng:

- **Git (Code):** Bn dũng **git push/pull** ng b code (võ d: **train.py**) vị cõc file **.dvc** siõu d liu (ch nng vị KB) liõn mỳ ch Git (nh GitHub, GitLab).

- **DVC (Data):** Bn dùng `dvc push/pull` ng b còc file d liu ln thc t (vò d: `model.pkl` 500MB) lỏn mt mỳ ch lu tr (Storage) riỏng bit (nh S3, Azure, Google Cloud, SSH).

file `model.pkl.dvc` (1KB) mị Git theo dủi ch lị mt "con tr" (pointer) tr n file `model.pkl` (500MB) thc s c DVC qun lý. iu nỳ gi cho kho Git ca bn luũn nh gn.

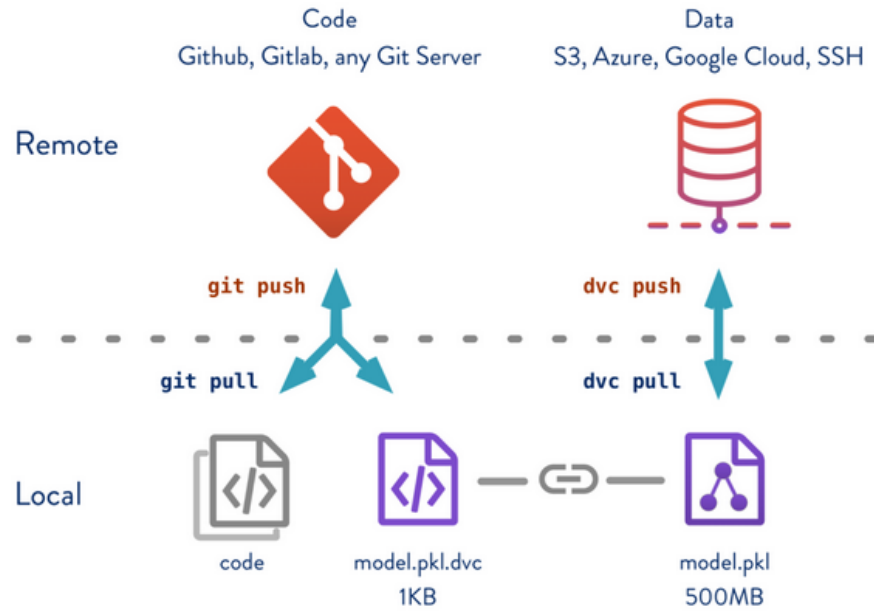


Figure 9: DVC vậi Git hot ng song song: Git qun lý Code, DVC qun lý Data

DVC có tính năng 1-1 so với Git và cú pháp (ví dụ: `dvc add`, `dvc push`, `dvc pull`), giúp bất kỳ ai quen thuộc với version control của git hiểu ngay khi sử dụng.

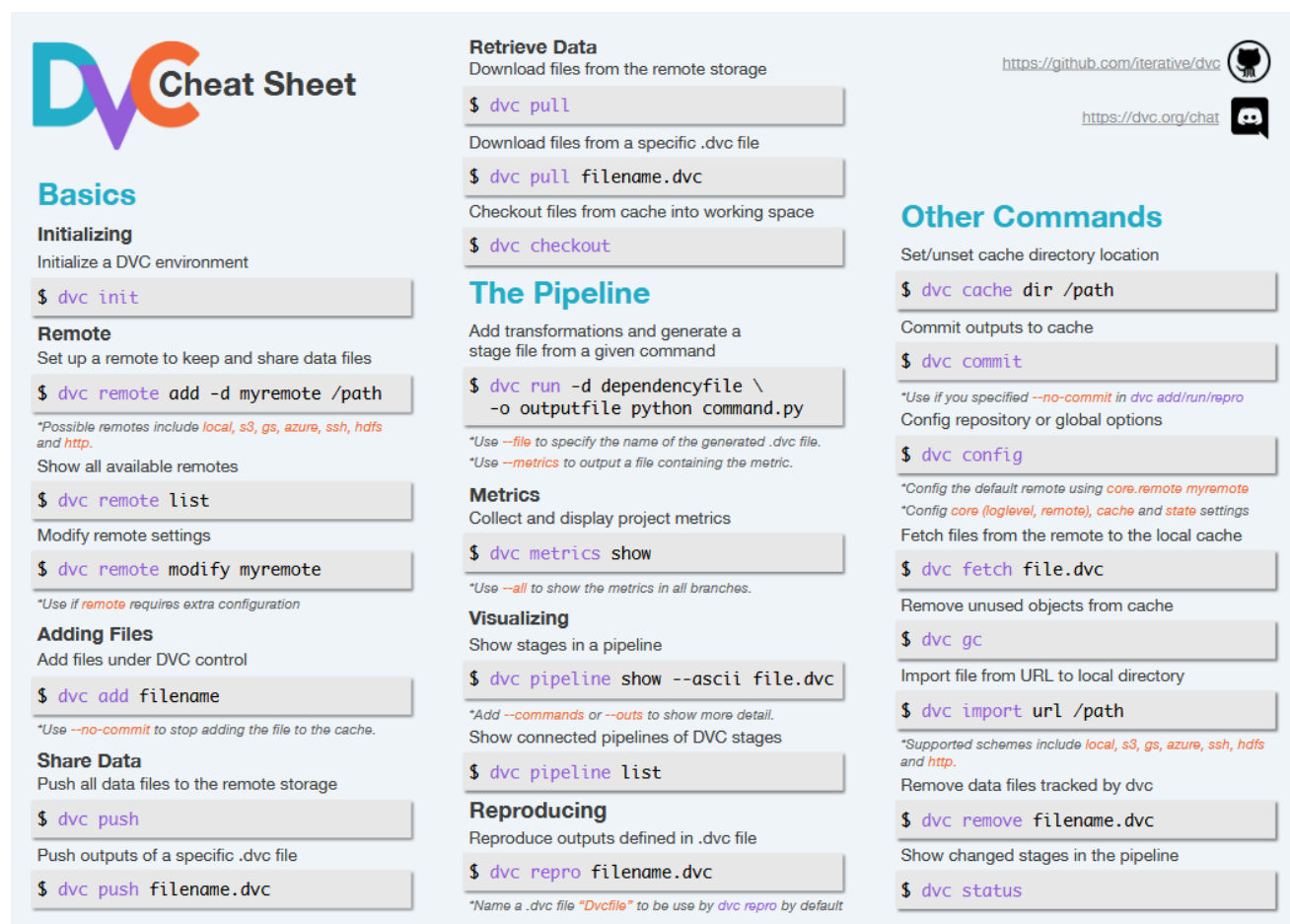


Figure 10: DVC Cheat Sheet cho thấy sự tương ứng giữa dòng lệnh Git

### 3.3 Quy trình DVC Pipeline hoàn chỉnh

Một DVC pipeline hoàn chỉnh cho phép bạn tự động hóa toàn bộ quy trình ML từ việc tạo mô hình (reproducibility). Bạn hãy thử cài đặt cấu hình (config) của pipeline từ file `dvc.yaml`.

#### 3.3.1 Định nghĩa Pipeline (dvc.yaml)

File `dvc.yaml` là nơi bạn định nghĩa tất cả các **stages** (giai đoạn) của quy trình ML. Mỗi stage giống như một bước trong "cổng thực" của bạn. Một stage thường bao gồm:

- **cmd**: Lệnh thực thi (ví dụ: `python train.py`).
- **deps**: Các file phụ thuộc (dependencies) của giai đoạn, ví dụ như code ('train.py') hoặc dữ liệu thô ('data/raw.csv').
- **params**: Các tham số (parameters), thông số định nghĩa trong file `params.yaml` (ví dụ: learning rate, số epochs).
- **outs**: Các file đầu ra (outputs) của mỗi stage này tạo ra, ví dụ như mô hình đã huấn luyện (`model.pkl`) hoặc dữ liệu đã xử lý.



### 3.3.2 Thực thi Pipeline (dvc repro)

Khi bạn chạy lệnh `dvc repro`, DVC sẽ thực hiện một việc rất thú vị minh: Nó kiểm tra file `dvc.yaml` và so sánh **hash** (mã chuỗi như danh duy nhất) của các file **deps** và **params** hiện tại và thông tin được lưu trong file `dvc.lock`.

- file `dvc.lock` lưu lại "dấu vân tay" (hash) của các file **deps**, **params**, và **outs** để bạn thấy chúng cũng được cập nhật.
- Nếu hash của bất kỳ file **deps** (ví dụ: bạn sửa code `train.py`) hoặc bất kỳ **params** nào thay đổi, DVC sẽ nhận ra stage đó là "lỗi thời" (outdated) và **chạy lại stage đó** bằng cách chạy các stage phụ thuộc vào nó.
- Nếu không có gì thay đổi, DVC sẽ không chạy gì cả, giúp tiết kiệm thời gian tính toán.

Để chờ đợi việc chạy DVC mà bạn không muốn chờ đợi vì code và dữ liệu quá lớn.

## 3.4 Cài đặt DVC trên máy tính

Để cài đặt những gì cần thiết để cài đặt DVC, bạn cần cài đặt (tức là cài đặt) hi quiz bài Th 5 Module 5 Week 1) trước khi đi vào thực hành.

### 3.4.1 Quy trình làm việc chung: DVC trước, Git sau

Để quy trình làm việc chung nhất. Khi bạn có một file dữ liệu lớn (ví dụ: `data.zip`):

1. **'dvc add data.zip'**: Lệnh này cho DVC biết theo dõi `data.zip`. DVC sao chép file này vào kho lưu trữ cục bộ (`.dvc/cache`) và tạo ra một file siêu dữ liệu (pointer) như tồn tại `data.zip.dvc`.
2. **'git add data.zip.dvc'**: Bạn dùng Git để theo dõi file pointer `.dvc` (chỉ vài KB), **KHÔNG BAO GIỜ** 'git add' file `data.zip` gốc.
3. **'git commit -m "Track new data"'**: Lưu lại trạng thái của file pointer.

Nếu bạn 'git add' file lớn trước, kho Git của bạn sẽ bị phình to vì DVC sẽ không thể quản lý file đó.

### 3.4.2 git push vs. dvc push

Để hiểu rõ hơn về việc cài đặt (như trong hình `git_v_dvc.png`):

- **'git push'**: Chỉ đẩy code ('.py') và các file siêu dữ liệu/con tr ('.dvc', 'dvc.yaml') lên máy chủ Git (như GitHub).
- **'dvc push'**: Đẩy các file dữ liệu lớn thực tế (mọi DVC đang theo dõi) từ cache cục bộ (`.dvc/cache`) lên kho lưu trữ từ xa (Remote Storage) như AWS S3, GCS, hoặc SSH.

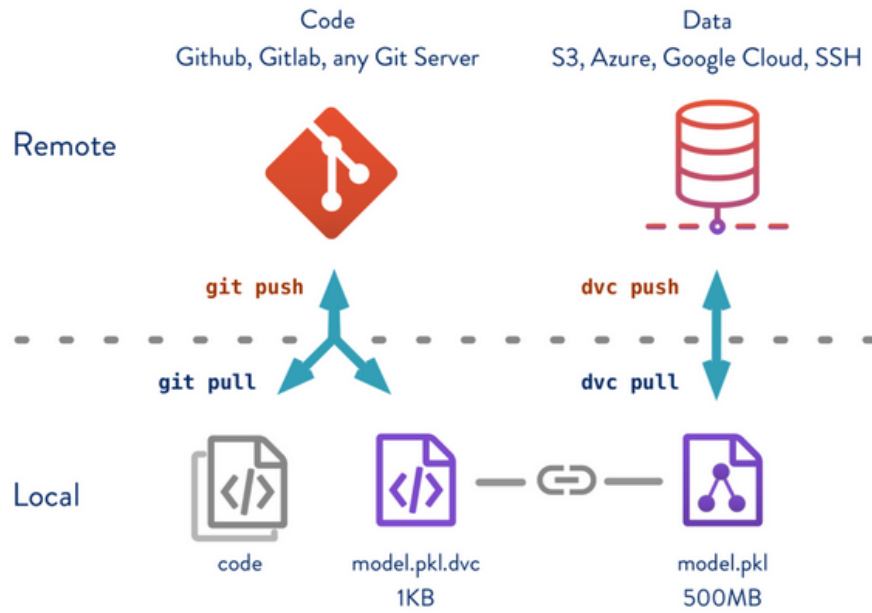


Figure 11: Git quản lý Code/Metadata, DVC quản lý Data thực tế

### 3.4.3 dvc pull vs. dvc checkout

Khi làm việc nhóm, bạn sẽ đứng hai lĩnh vực:

- **‘dvc pull’**: Tải dữ liệu từ Remote Storage (ví dụ: S3) về kho lưu trữ cục bộ (Local Cache) của bạn (thư mục `.dvc/cache`).
- **‘dvc checkout’**: Ngăn chặn dữ liệu từ Local Cache ra thành môi trường làm việc (workspace) của bạn. Lĩnh vực này chỉ có file `.dvc` trong workspace và tạo liên kết (symlink) đến file thực tế trong cache. Bạn thường chỉ cần lĩnh vực này sau khi `git checkout` một branch mới.

### 3.4.4 Yêu cầu tối thiểu cho DVC Pipeline

Trước khi có thể chạy `dvc repro`, bạn cần làm theo:

1. Khởi tạo kho Git (`git init`).
2. Cài đặt các thư viện cần thiết (ví dụ: `pip install dvc dvc-s3`).
3. Định nghĩa các stage trong file `dvc.yaml`.
4. (Tùy chọn) Thêm vị trí kho lưu trữ AWS bằng lệnh `dvc remote add`, ví dụ: `dvc remote add -d my-s3-storage s3://my-bucket/dvc-storage`. [Hãy đọc setup AWS của TA trên Notion](#)

## Phần 4: Case Study: Triển khai DVC cho Dataset MNIST

Chúng ta sẽ thực hiện một case study từng bước về DVC và Git để hiểu rõ hơn về cách quản lý các phiên bản thực nghiệm (dữ liệu và mô hình).

## 4.1 Bc 1: Thit lp D òn vj Git

u tĩn, chĩng ta to th mc d òn, cu trũc th mc con, mũi trng conda, vj quan trng nhĩ lĩ khi to kho Git.

```
1 $ # 1. Create project directory and basic structure
2 $ mkdir dvc-mnist-demo
3 $ cd dvc-mnist-demo
4 $ mkdir data/raw models scripts
5
6 $ # 2. Create conda environment and install libraries
7 $ conda create -n dvc_mnist python=3.11
8 $ conda activate dvc_mnist
9 $ pip install -r requirements.txt
10
11 $ # 3. Initialize Git and make the first commit
12 $ git init
13 $ git add .
14 $ git commit -m "Init project"
```

### Step 1: Set Up Project

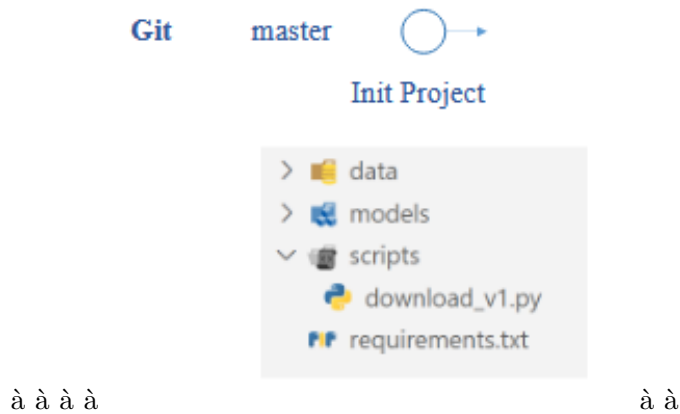


Figure 12: Trng thĩi Git vj cu trũc file ban u

## 4.2 Bc 2: Tĩ Dataset V1 vj Theo dũĩ bng DVC

Tip theo, chĩng ta tĩ phiĩn bn u tĩn ca dataset (Full MNIST) vj dũĩng DVC bt u theo dũĩ nũ.

```
1 $ # 1. Download data (60000 samples)
2 $ python scripts/download_v1.py
3
4 $ # 2. Initialize DVC in the project
5 $ dvc init
6
7 $ # 3. Ask DVC to track the large data files
8 $ dvc add data/raw/x_train_v1.npy data/raw/y_train_v1.npy data/raw/x_test.npy data/
   raw/y_test.npy
```

Lnh dvc add s to ra cĩc tp .dvc (con tr) siĩu d liũ vj t ng thĩĩm cĩc tp .npy ln vĩo .gitignore.

### 4.3 Bc 3: Commit Phiên bản D liệu V1

Giờ chúng ta commit các tệp .dvc (con trỏ) vào Git để lưu lại "phiên bản" dữ liệu này.

```
1 $ # Add .dvc files (pointers) and the .gitignore file updated by DVC
2 $ git add data/raw/.gitignore data/raw/x_train\_v1.npy.dvc data/raw/y_train\_v1.npy.
   dvc data/raw/x_test.npy.dvc data/raw/y_test.npy.dvc
3
4 $ # Commit to finalize Version V1
5 $ git commit -m "Version 1: Full MNIST dataset"
```

#### Step 3: Commit to Git

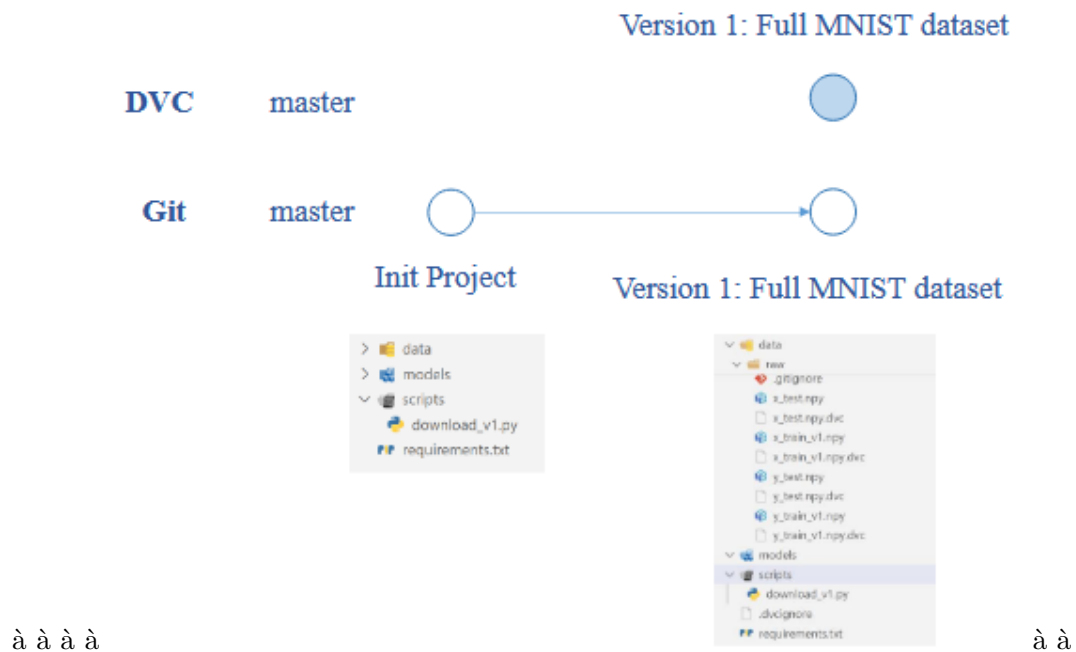


Figure 13: Lịch sử Git và DVC song song sau khi commit dữ liệu V1

### 4.4 Bc 4: Huấn luyện và Theo dõi Mô hình V1

Với dữ liệu V1, giờ chúng ta huấn luyện mô hình và tiến hành theo dõi các tệp mô hình và chỉ số (metrics) của nó.

```
1 $ # 1. (Optional) Create symbolic link for train.py script to read data
2 $ cd data/raw
3 $ mklink x_train.npy x_train\_v1.npy
4 $ mklink y_train.npy y_train\_v1.npy
5 $ cd ../../
6
7 $ # 2. Run training (on 60000 samples, achieved 0.9319 accuracy)
8 $ python scripts/train.py
9
10 $ # 3. Ask DVC to track output files (model and metrics)
11 $ dvc add models/model.npy
12 $ dvc add models/metrics.json
13
14 $ # 4. Commit V1 model's .dvc pointers to Git
```

```

15 $ git add .
16 $ git commit -m "Model_v1: Full dataset accuracy"

```

#### Step 4: Training

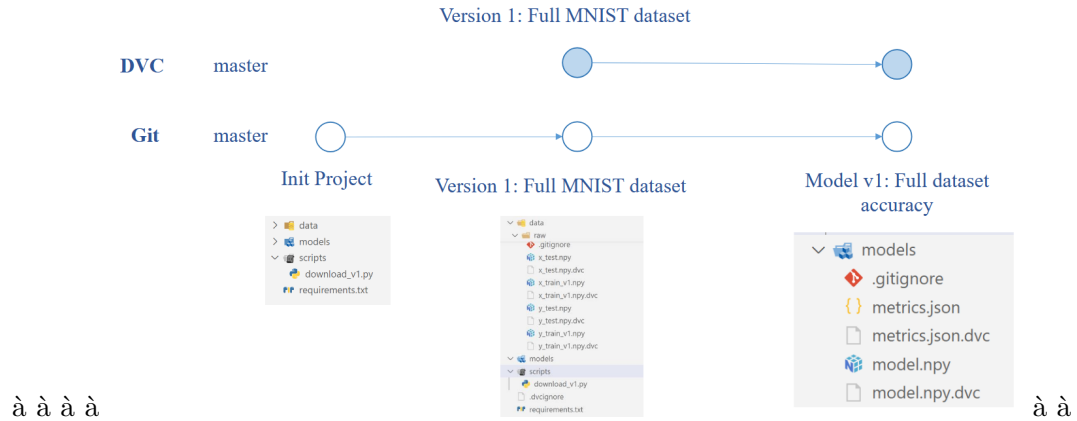


Figure 14: Toj n b lch s Git vj DVC sau khi hun luy n vj commit Mũ hnh V1

#### 4.5 Bc 5: To Phiõn bn D liu V2

Bóy gi, chũng ta gi lp mt th nghim mi bng cõch to phiõn bn d liu th hai (ch 1000 mu).

```

1 $ # 1. Run download\_v2.py script to create V2 data
2 $ python scripts/download\_v2.py
3
4 $ # 2. Ask DVC to track the new V2 data files
5 $ dvc add data/raw/x_train\_v2.npy
6 $ dvc add data/raw/y_train\_v2.npy
7
8 $ # 3. Commit V2's .dvc pointers to Git
9 $ git add .
10 $ git commit -m "Dataset_V2"

```

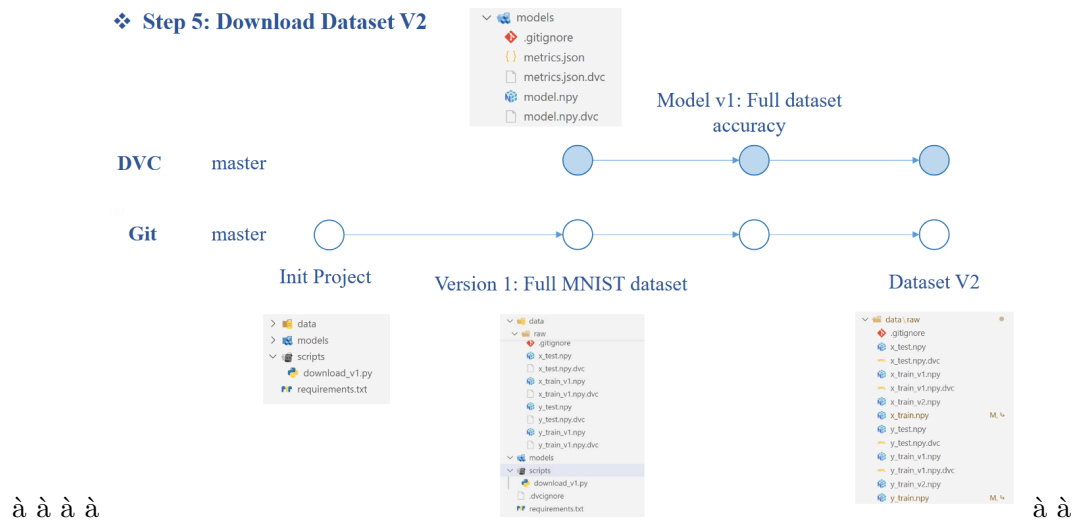


Figure 15: Lch s Git vj DVC sau khi thõm "Dataset V2"

## 4.6 Bc 6: Huynh luyt Mũ hnh V2

Chũng ta lp li quy trnh huyn luyt, nhng ln nỳ s dng d liu V2 (bng cõch thay i symbolic link) to ra mũ hnh V2.

```
1 $ # 1. Update symbolic link to point to V2 data
2 $ cd data/raw
3 $ del x_train.npy y_train.npy
4 $ mklink x_train.npy x_train\v2.npy
5 $ mklink y_train.npy y_train\v2.npy
6 $ cd ../../
7
8 $ # 2. Run training (on 1000 samples, achieved 0.8047 accuracy)
9 $ python scripts/train.py
```

## 4.7 Bc 7: Theo dũi Mũ hnh V2

Gi chũng ta theo dũi cõc tp mũ hnh vớ ch s V2 mĩ.

```
1 $ # 1. Ask DVC to track V2 model and metrics files
2 $ dvc add models/model.npy
3 $ dvc add models/metrics.json
4
5 $ # 2. Commit V2 model's .dvc pointers
6 $ git add .
7 $ git commit -m "Model_v2: Small_dataset_accuracy"
```

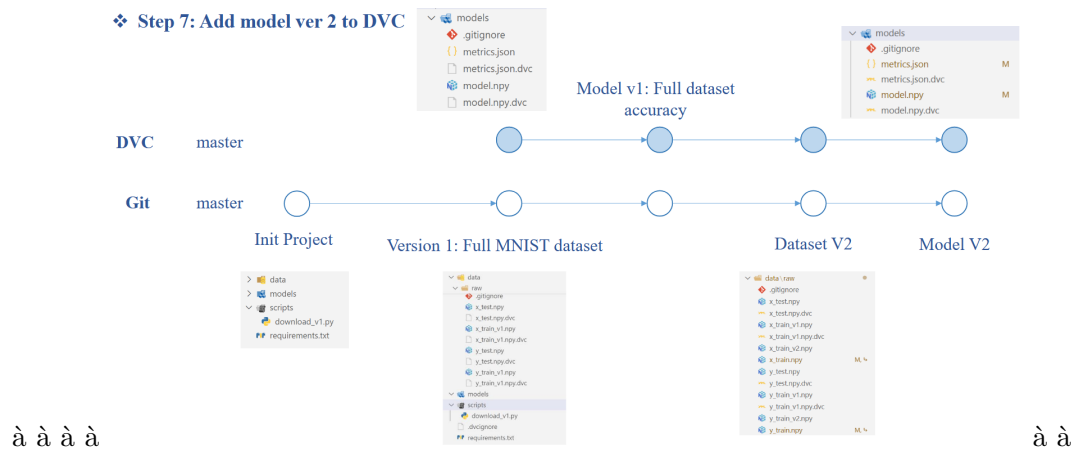


Figure 16: Lch s Git/DVC hoĩn chnh vớ 2 phiĩn bn d liu vớ 2 phiĩn bn mũ hnh

## 4.8 Bc 8: Cu hnh Kho lu tr (Storage)

Cõc tp d liu ln thcs nm trong cache (.dvc/cache). chia s chũng, ta cn thit lp mt kho lu tr t xa (remote storage).

### 4.8.1 S dng Local Storage (Lu tr Cc b)

ý lĩ cõch n gin chia s cache trong cĩng mt mỳi hoc qua mng ni b.

```
1 $ # 1. Create a directory outside the project to act as "storage"
2 $ mkdir dvc_storage
3 $ # 2. Add it as the "localremote" (save configuration to .dvc/config)
4 $ dvc remote add -d localremote ./dvc_storage
```

```

5 $ # 3. Push data from cache (.dvc/cache) to localremote
6 $ dvc push

```

## 4.8.2 S dng Cloud Storage (vở d: AWS S3)

óy lị còch lịm ph bin nht khi lịm vic nhúm.

```

1 $ # 1. Install the S3 support library
2 $ pip install dvc-s3
3 $ # 2. Add the S3 bucket as the "mys3" remote
4 $ dvc remote add -d mys3 s3://dvc-mnist-demo-bucket/data
5 $ # 3. Push data from cache to S3 (the default remote)
6 $ dvc push

```

## 4.9 Bc 9: Kim tra Chuyn i Phiổn bn

óy lị sc mnht ln nht ca DVC. Chũng ta cú th quay li bt k th nghim nịo trong quỏ kh. Vở d, quay li commit ca Mũ hnh V1:

```

1 $ # 1. Find the commit_id of V1 (e.g., "Model v1: Full dataset accuracy")
2 $ git log
3
4 $ # 2. Revert to the code state of that commit
5 $ git checkout <commit_id_cua_V1>
6
7 $ # 3. Ask DVC to sync the corresponding data/model for that commit
8 $ dvc checkout
9
10 $ # 4. Check (will see V1 shape is 60000)
11 $ python -c "import numpy as np; print('V1 Data shape:', np.load('data/raw/x_train.npy').shape)"

```

Nu bn git checkout master vị dvc checkout mt ln na, d liu s quay v V2 (1000 mu).

## 4.10 Bc 10: Push vị Clone (Lịm vic nhúm)

Quy trnh lịm vic nhúm in hnh:

```

1 $ # --- Person A (Pushing the project) ---
2 $ # 1. Push code and .dvc pointers to GitHub
3 $ git remote add origin <your-repository>
4 $ git push origin master
5 $ # 2. Push the actual data to Cloud Storage
6 $ dvc push
7
8 $ # --- Person B (Cloning the project) ---
9 $ # 1. Clone code and .dvc pointers from GitHub
10 $ git clone <your-repository>
11 $ # 2. Connect to Cloud Storage
12 $ dvc remote add -d mys3 s3://dvc-mnist-demo-bucket/data
13 $ # 3. Pull the actual data from Cloud Storage to cache
14 $ dvc pull

```

## 4.11 Bc 11: Tng kt Thao tồc DVC c bn

Workflow trổn minh ha còc lnh DVC c bn bn s s dng hịng ngiy:

### Step 11: DVC Operations

# Push all code to GitHub

```
$ dvc list .
$ dvc status
$ dvc pull
```

```
(dvc_mnist) D:\OneDrive\TA AIO\AI02025\Module 05\dvc-mnist-demo>dvc list .
.dvcignore
data
models
requirements.txt
scripts
```

# Basic DVC workflow

```
$ dvc add data/raw/file.npy # Track data file
$ dvc checkout # Switch data versions
$ dvc push # Send to remote
$ dvc pull # Get from remote
$ dvc status # Check changes
```

à à

à à

Figure 17: Cờ nh DVC workflow c bn

```
1 $ # Ask DVC to track a file. Note: Track data file
2 $ dvc add data/raw/file.npy
3
4 $ # Sync data from cache to working directory (when checking out Git). Note: Switch
   data versions
5 $ dvc checkout
6
7 $ # Push data (large files) from cache to remote storage (S3, GCS...). Note: Send to
   remote
8 $ dvc push
9
10 $ # Pull data (large files) from remote storage to cache. Note: Get from remote
11 $ dvc pull
12
13 $ # Check status of data files compared to Git commit. Note: Check changes
14 $ dvc status
15
16 $ # List files currently tracked by DVC in the project
17 $ dvc list .
```

## Phn 5: T ng húa Pipelines vớ Cờ khò nim Versioning

Mt trong nhng tồnh nng mnh m nht ca DVC lợ kh nng t ng húa toỏn b quy trỏnh Machine Learning (ML pipeline) thũng qua cờ tp cu hỏnh (Configure File).

Tp cu hỏnh chờnh lợ `dvc.yaml`. Bn cú th coi nú nh mt "bn cũng thc" hay mt "bn k hoch chỉ tit" cho đ ỏn ca bn. Nú lợ mt tp vn bn n gin mợ con ngợ cú th c c, thng nm th mc gc (root) ca đ ỏn. Tp nợ nh ngha tt c cờ **stages** (giai ỏn) trong quy trỏnh ca bn, vớ d: "bc 1: tỉ đ liu", "bc 2: x lý đ liu", "bc 3: hun luy nũ hỏnh".

Mc ờch ca nú lợ gợi quy t vn "phi chy th cũng" (manual work) vớ m bo **kh nng tời lp (reproducibility)**. Thay vớ phi nh chy 5 tp Python theo ũng th t, bn ch cn nh ngha cũng mt ln trong `dvc.yaml`. Sau ú, DVC s t ng bit phi chy gợ, theo th t nợ, vớ quan trng nht: ch chy li nhng bc b nh hng khi code hoc đ liu ca bn thay i, gợp tit kim rt nhiu th gian.



## 5.1 DVC Automation Pipeline (Quy trình Tự động hóa DVC)

Hãy xem cách tạo file `dvc.yaml` như sau để pipeline.

### Configure File: `dvc.yaml`



```
stages:
  download_v1:
    cmd: python scripts/download_v1.py
    deps:
      - scripts/download_v1.py
    outs:
      - data/raw/x_train_v1.npy
      - data/raw/y_train_v1.npy
      - data/raw/x_test.npy
      - data/raw/y_test.npy

  download_v2:
    cmd: python scripts/download_v2.py
    deps:
      - scripts/download_v2.py
      - data/raw/x_train_v1.npy
      - data/raw/y_train_v1.npy
    outs:
      - data/raw/x_train_v2.npy
      - data/raw/y_train_v2.npy

train_v1:
  cmd: python scripts/train.py --version v1
  deps:
    - scripts/train.py
    - data/raw/x_train_v1.npy
    - data/raw/y_train_v1.npy
    - data/raw/x_test.npy
    - data/raw/y_test.npy
  outs:
    - models/model_v1.npy
  metrics:
    - models/metrics_v1.json:
        cache: false

train_v2:
  cmd: python scripts/train.py --version v2
  deps:
    - scripts/train.py
    - data/raw/x_train_v2.npy
    - data/raw/y_train_v2.npy
    - data/raw/x_test.npy
    - data/raw/y_test.npy
  outs:
    - models/model_v2.npy
  metrics:
    - models/metrics_v2.json:
        cache: false
```

Figure 18: Cấu hình các giai đoạn (stages) trong `dvc.yaml`

Trong file `dvc.yaml`, mỗi stage được định nghĩa trong `stages`:

- **‘stages:’**: Khai báo tổng thể danh sách các stage.
- **‘download\_v1:’**: Tên của một stage.
- **‘cmd:’**: Lệnh (command) sẽ thực thi khi chạy stage này (ví dụ: `python scripts/download_v1.py`).
- **‘deps:’**: Các tệp phụ thuộc (dependencies). DVC sẽ theo dõi các tệp này. Nếu tệp `scripts/download_v1.py` thay đổi, DVC sẽ bắt đầu stage này "liên tiếp" với các tệp phụ thuộc.
- **‘outs:’**: Các tệp đầu ra (outputs). DVC sẽ theo dõi các tệp này (ví dụ: `data/raw/x_train_v1.npy`). DVC sẽ theo dõi (giống như `dvc add`) các tệp này.

Bạn có thể thấy các stage `train_v1` và `train_v2` phụ thuộc vào script (`train.py`) và dữ liệu đầu vào (ví dụ: `data/raw/x_train_v1.npy`). Điều này tạo ra một chuỗi liên kết, hay một **DAG (Directed Acyclic Graph)**, mà DVC hiểu rằng phải chạy `download_v1` trước khi chạy `train_v1`.

### 5.1.1 Sử dụng `params.yaml` trong Pipeline

Vì sao chúng ta cần các stage `train_v1` và `train_v2` (như trong hình trên) rất dễ hiểu. Một cách tốt hơn là sử dụng tệp `params.yaml` để lưu trữ các tham số (hyperparameters).

## Configure File: params.yaml



Figure 19: Kt hp dvc.yaml (phi) vị params.yaml (trời)

Thay vớ nh ngha hai stage riểng bit, chũng ta s dng "templating" (to mu):

1. '**params.yaml**': Chũng ta nh ngha mt bin `data_version`: `["v1", "v2"]`.
2. '**dvc.yaml**': Chũng ta to mt stage `train_${data_version}` (tổn stage ng).
3. '**foreach**': DVC s lp qua tng mc (item) trong `data_version`.
4. '**\$item**': Bin gì ch nịy s c thay th bng "v1" vị "v2" khi chy.

Vì cồh nịy, bn ch cn nh ngha stage `train` mt ln. Nũ sau nịy bn mun thỏm "v3", bn ch cn thỏm "v3" vị ỏ tp `params.yaml` mị khũng cn sa `dvc.yaml`.

## 5.2 Ý tng ca Versioning (Phiổn bn húa)

Khòì nìm ct lủi ca DVC lị m rng Git workflow (quy trỡnh lịm vic ca Git) cho d liu vị cồc th nghim (experiments). Cồc hot ng tng t git, mị nhồnh tng ng vị 1 hng phồtrìn.

## Data Version Control (DVC)

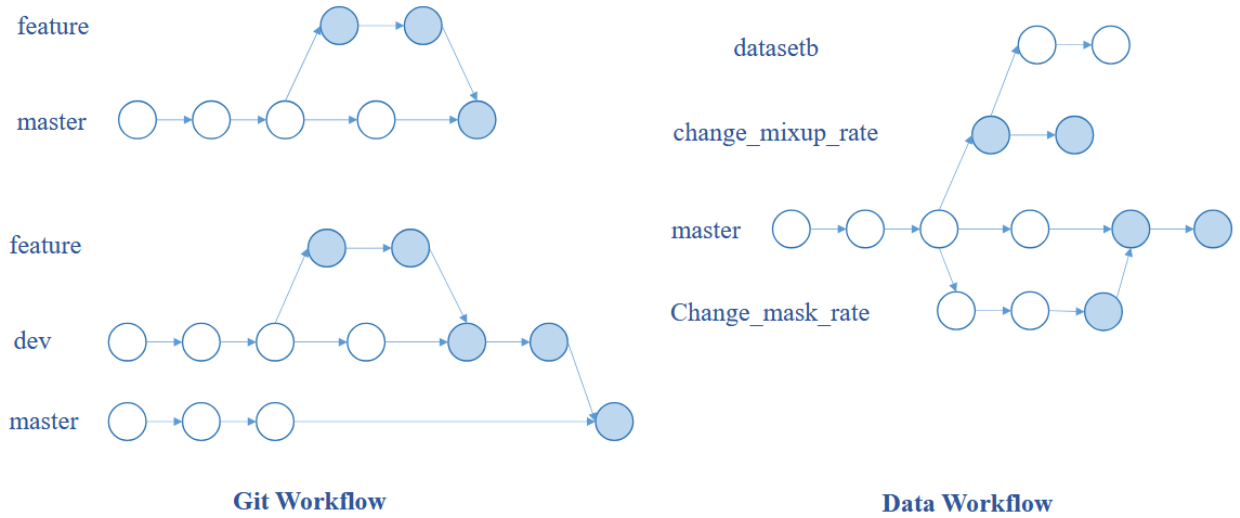


Figure 20: So sánh Git Workflow (trò) và Data Workflow (phi)

- **Git Workflow (Bổn trò):** Trong phát triển phần mềm, khi muốn phát triển một tính năng mới (feature), bạn tạo một **nhánh (branch)** mới từ **master**, thử nghiệm code (còn commit màu xanh), và khi hoàn thiện, bạn gộp (merge) nó trở lại.
- **Data Workflow (Bổn phi):** Chúng ta áp dụng ý tưởng tương tự cho các thí nghiệm ML.
  - Nhánh **master** là mô hình chờn (production model) của bạn.
  - Khi bạn muốn thử một ý tưởng mới, hoặc thay đổi tham số **mask\_rate**, bạn tạo một nhánh Git mới từ nhánh **Change\_mask\_rate**.
  - Trở về nhánh này, bạn thay đổi tệp **params.yaml**, sau đó chỉ **dvc repro**. DVC sẽ tạo ra mô hình và các số liệu, sau đó bạn **git commit** các thay đổi (ví dụ: **dvc.lock**).
  - Tương tự, bạn có thể tạo nhánh **change\_mixup\_rate** hoặc nhánh **datasetb** để thử nghiệm dữ liệu mới.

Bằng cách này, mô hình Git rất phù hợp cho các thí nghiệm ML hoàn chỉnh và có thể tái lập. Bạn có thể dễ dàng chuyển đổi giữa các thí nghiệm (**git checkout**) và so sánh kết quả (**dvc metrics diff**) mà không làm ảnh hưởng đến nhánh **master** chờn.

## References

[Ite25] Iterative. *Data Version Control về DVC*. <https://dvc.org/>. Accessed: 2025-10-18. 2025.