Data Version Control

DVC Pipeline Configuration Guide

DVC

Complete Reference for ML Pipeline Automation

Version Control for Data & Models

Sujil S

sujil9480@gmail.com

December 25, 2025

Contents

1 Introduction to DVC

1.1 What is DVC?

DVC (Data Version Control) is an open-source version control system for machine learning projects that extends Git's capabilities to handle large datasets, models, and ML workflows.

1.1.1 Core Features

- **Large File Management**: Efficiently version datasets >100MB
- **Pipeline Automation**: Reproducible ML workflows
- **Experiment Tracking**: Compare metrics and parameters
- **Model Versioning**: Track model evolution over time
- **Remote Storage**: Works with S3, GCS, Azure, SSH
- **Git Integration**: Seamless integration with Git workflow

1.2 Why Use DVC?

1.2.1 Traditional Git Limitations

Problems with Git for ML Projects:

- Cannot handle files >100MB efficiently
- Not optimized for binary files (models, datasets)
- No built-in experiment tracking
- Lacks pipeline automation
- Difficult to reproduce ML experiments
- Poor performance with large repositories

1.2.2 DVC Solutions

How DVC Solves These Problems:

- **Efficient Storage**: Links to large files without storing in Git
- **Pipeline Automation**: Define workflows with `dvc.yaml`
- **Experiment Tracking**: Built-in metrics and parameter comparison
- **Reproducibility**: Guaranteed consistent results with `dvc.lock`
- **Collaboration**: Share data seamlessly via remote storage
- **Storage Agnostic**: Compatible with any cloud provider

1.3 DVC Architecture

1.3.1 Core Components

1. **.dvc files**: Lightweight pointers to actual data
2. **dvc.yaml**: Pipeline definition and stage configuration
3. **dvc.lock**: Lock file with checksums for reproducibility
4. **params.yaml**: Centralized parameter management
5. **metrics/**: JSON/YAML files with evaluation metrics
6. **.dvc/cache**: Local cache for tracked files
7. **Remote Storage**: Cloud storage for team collaboration

1.3.2 Workflow Comparison

Git Workflow	**DVC Workflow**
Tracks source code	Tracks data and models
Optimized for text files	Optimized for large binary files
Stores in `.git/` directory	Stores in `.dvc/cache`
`git commit, push, pull`	`dvc add, push, pull`
Remote: GitHub, GitLab	Remote: S3, GCS, Azure
Files tracked directly	Files tracked via pointers

1.4 Getting Started

1.4.1 Installation

```
# Install DVC with pip
pip install dvc

# Install with specific remote support
pip install dvc[s3]      # AWS S3
pip install dvc[gs]      # Google Cloud Storage
pip install dvc[azure]   # Azure Blob Storage
pip install dvc[ssh]     # SSH remote
pip install dvc[all]     # All remotes

# Verify installation
dvc version
```

1.4.2 Project Initialization

```
# Initialize Git repository
git init

# Initialize DVC
dvc init

# Commit DVC configuration
```

```
git add .dvc .gitignore
git commit -m "Initialize DVC"
```

2 Understanding dvc.yaml

2.1 What is dvc.yaml?

dvc.yaml is the central pipeline configuration file that defines your entire ML workflow. It describes stages, dependencies, commands, outputs, parameters, and metrics in a declarative YAML format.

2.2 Basic Structure

2.2.1 Minimal Pipeline

```yaml
stages:
 stage_name:
  cmd: python script.py
  deps:
   - input_file.csv
  outs:
   - output_file.csv
```

2.2.2 Complete Stage Definition

```yaml
stages:
 training:
  cmd: python train.py --epochs 100
  wdir: src/            # Working directory
  deps:
   - data/processed/train.csv   # Data dependencies
   - src/train.py         # Code dependencies
   - src/models/architecture.py  # Module dependencies
  params:
   - training.learning_rate    # Parameter references
   - training.batch_size
   - model.architecture
  outs:
   - models/model.pkl       # Model outputs
   - models/weights.h5
  metrics:
   - metrics/scores.json:     # Metrics (not cached)
     cache: false
  plots:
   - plots/training_curve.csv:   # Plot data
     cache: false
     x: epoch
     y: loss
  frozen: false           # Allow automatic execution
```

2.3 Key Benefits

Why Use dvc.yaml?

- **Reproducibility**: Execute exact same workflow every time
- **Automation**: Run entire pipeline with single command
- **Dependency Tracking**: Only re-run stages with changes
- **Version Control**: Track pipeline evolution in Git
- **Documentation**: Self-documenting workflow
- **Collaboration**: Share workflows with team members

3 Core Components

3.1 Stages

Stages are the fundamental building blocks representing discrete steps in your ML pipeline.

3.1.1 Stage Characteristics

- Each stage has a unique identifier name
- Stages execute in dependency order (DAG)
- Outputs are cached to avoid redundant computation
- Changes in dependencies trigger re-execution
- Multiple stages can run in parallel if independent

3.1.2 Simple Stage Example

```
stages:
 preprocess:
  cmd: python src/preprocess.py
  deps:
   - data/raw/dataset.csv
   - src/preprocess.py
  outs:
   - data/processed/train.csv
   - data/processed/test.csv
```

3.2 Commands (cmd)

The **cmd** field specifies the shell command to execute.

3.2.1 Command Variations

```
# Simple command
stages:
 train:
  cmd: python train.py

# Command with arguments
stages:
 train:
  cmd: python train.py --epochs 100 --lr 0.001

# Multi-line command
stages:
 pipeline:
  cmd: >
   python preprocess.py &&
   python train.py &&
   python evaluate.py

# Command with environment variables
```

```
stages:
 gpu_train:
  cmd: CUDA_VISIBLE_DEVICES=0 python train.py

# Command with shell features
stages:
 batch_process:
  cmd: for file in data/*.csv; do python process.py $file; done
```

3.3 Dependencies (deps)

Dependencies are files or directories that trigger stage re-execution when modified.

3.3.1 Dependency Types

```
stages:
 train:
  cmd: python train.py
  deps:
   # Data dependencies
   - data/processed/train.csv
   - data/processed/validation.csv

   # Code dependencies
   - src/train.py
   - src/models/neural_network.py
   - src/utils/data_loader.py

   # Configuration files
   - configs/model_config.yaml

   # Pre-trained models
   - models/pretrained/base_model.pkl

   # Directory dependencies
   - data/images/
```

Important: Always include ALL files that affect stage output. Missing dependencies can lead to inconsistent results and break reproducibility.

3.4 Outputs (outs)

Outputs are files or directories produced by a stage.

3.4.1 Output Configuration

```
stages:
 train:
  cmd: python train.py
  outs:
   # Simple outputs (cached by default)
   - models/model.pkl
   - models/weights.h5
```

```

    # Outputs with custom settings
    - models/final_model.pkl:
      cache: true         # Cache this output
      persist: false      # Remove when not current

    - logs/training.log:
      cache: false        # Don't cache logs

    - checkpoints/:
      cache: true
      persist: true       # Keep even when not current
```

3.4.2 Output Options

Option	Description
`cache`	Cache output (default: `true`)
`persist`	Keep in workspace when not current (default: `false`)
`checkpoint`	Mark as ML checkpoint for experiments
`desc`	Human-readable description

3.5 Parameters (params)

Parameters reference values from external files (typically `params.yaml`).

3.5.1 Parameter Usage

```
# dvc.yaml
stages:
 train:
  cmd: python train.py
  params:
   - training.epochs        # Specific parameter
   - training.learning_rate
   - training.batch_size
   - model             # Entire section
   - optimizer.type
   - optimizer.beta1
```

```
# params.yaml
training:
 epochs: 100
 learning_rate: 0.001
 batch_size: 32
 early_stopping: true

model:
 architecture: resnet50
 layers: 18
 dropout: 0.5
 activation: relu

optimizer:
 type: adam
 beta1: 0.9
```

```
  beta2: 0.999
```

3.5.2 Multiple Parameter Files

```
stages:
 train:
  cmd: python train.py
  params:
   - params.yaml:
     - training
     - model
   - configs/advanced.yaml:
     - augmentation
     - preprocessing
```

3.6 Metrics

Metrics are evaluation results stored in JSON, YAML, CSV, or TSV format.

3.6.1 Metric Definition

```
stages:
 evaluate:
  cmd: python evaluate.py
  deps:
   - data/test.csv
   - models/model.pkl
  metrics:
   - metrics/test_scores.json:
     cache: false       # NEVER cache metrics
   - metrics/confusion_matrix.json:
     cache: false
```

3.6.2 Metric File Format

```
{
  "accuracy": 0.9542,
  "precision": 0.9321,
  "recall": 0.9456,
  "f1_score": 0.9388,
  "auc_roc": 0.9782,
  "confusion_matrix": {
    "true_positive": 850,
    "false_positive": 45,
    "true_negative": 920,
    "false_negative": 35
  }
}
```

3.6.3 Viewing Metrics

```
# Show current metrics
dvc metrics show

# Compare with another branch
dvc metrics diff main

# Compare with previous commit
dvc metrics diff HEAD~1

# Show all experiments
dvc metrics show -R
```

3.7 Plots

Plots are data files for visualization (CSV, JSON, YAML).

3.7.1 Plot Configuration

```yaml
stages:
 evaluate:
  cmd: python evaluate.py
  plots:
   # Simple plot
   - plots/training_history.csv:
     cache: false

   # Customized plot
   - plots/roc_curve.json:
     cache: false
     x: fpr
     y: tpr
     title: "ROC Curve"
     x_label: "False Positive Rate"
     y_label: "True Positive Rate"

   # Multi-series plot
   - plots/metrics.csv:
     x: epoch
     y:
      plots/metrics.csv: [loss, val_loss]
     title: "Training vs Validation Loss"
```

3.7.2 Plot Data Format

```
# plots/training_history.csv
epoch,loss,accuracy,val_loss,val_accuracy
1,0.693,0.501,0.685,0.515
2,0.612,0.653,0.598,0.672
3,0.523,0.745,0.501,0.758
4,0.445,0.812,0.432,0.825
5,0.389,0.856,0.398,0.861
```

4 Advanced Pipeline Features

4.1 Working Directory (wdir)

Execute stages in a specific working directory.

```
stages:
 notebook_analysis:
  cmd: jupyter nbconvert --execute analysis.ipynb
  wdir: notebooks/
  deps:
   - notebooks/analysis.ipynb
   - data/input.csv
  outs:
   - notebooks/output.html
   - notebooks/results.csv
```

4.2 Frozen Stages

Prevent automatic execution of specific stages.

```
stages:
 expensive_preprocessing:
  cmd: python expensive_process.py
  frozen: true          # Skip in dvc repro
  deps:
   - data/raw/large_dataset.csv
  outs:
   - data/processed/features.pkl
```

Use Cases for Frozen Stages:

- Computationally expensive operations
- Stages requiring manual intervention
- Optional pipeline branches
- Data collection/download stages

4.3 Always Changed Stages

Force re-execution regardless of dependencies.

```
stages:
 fetch_data:
  cmd: python fetch_latest_data.py
  always_changed: true      # Always runs
  outs:
   - data/latest/dataset.csv
   - data/latest/metadata.json
```

4.4 Foreach Loops

Execute the same stage multiple times with different parameters.

4.4.1 Simple Foreach

```yaml
stages:
 train_models:
  foreach:
   - logistic_regression
   - random_forest
   - gradient_boosting
   - neural_network
   - svm
  do:
   cmd: python train.py --model ${item}
   deps:
    - data/train.csv
    - src/train.py
   params:
    - models.${item}
   outs:
    - models/${item}/model.pkl
   metrics:
    - metrics/${item}_scores.json:
      cache: false
```

4.4.2 Dictionary Foreach

```yaml
stages:
 process_datasets:
  foreach:
   train: data/raw/train.csv
   validation: data/raw/val.csv
   test: data/raw/test.csv
  do:
   cmd: python process.py --input ${item} --output data/processed/${key}.csv
   deps:
    - ${item}
    - src/process.py
   outs:
    - data/processed/${key}.csv
```

4.4.3 Matrix Foreach (Hyperparameter Grid)

```yaml
stages:
 grid_search:
  foreach:
   lr_0001_bs16:
    learning_rate: 0.001
    batch_size: 16
   lr_0001_bs32:
    learning_rate: 0.001
    batch_size: 32
   lr_001_bs16:
    learning_rate: 0.01
    batch_size: 16
   lr_001_bs32:
    learning_rate: 0.01
```

```
      batch_size: 32
    do:
     cmd: >
      python train.py
      --lr ${item.learning_rate}
      --batch ${item.batch_size}
     deps:
      - data/train.csv
     outs:
      - models/${key}_model.pkl
     metrics:
      - metrics/${key}_scores.json:
        cache: false
```

4.5 Variables and Templating

Define reusable variables for cleaner configuration.

```
vars:
 - base_dir: .
 - data_dir: data
 - models_dir: models
 - metrics_dir: metrics
 - src_dir: src
 - batch_size: 32
 - learning_rate: 0.001
 - random_seed: 42

stages:
 preprocess:
  cmd: python ${src_dir}/preprocess.py --seed ${random_seed}
  deps:
   - ${data_dir}/raw/dataset.csv
   - ${src_dir}/preprocess.py
  outs:
   - ${data_dir}/processed/train.csv
   - ${data_dir}/processed/test.csv

 train:
  cmd: >
   python ${src_dir}/train.py
   --batch-size ${batch_size}
   --lr ${learning_rate}
   --seed ${random_seed}
  deps:
   - ${data_dir}/processed/train.csv
   - ${src_dir}/train.py
  outs:
   - ${models_dir}/model.pkl
  metrics:
   - ${metrics_dir}/scores.json:
     cache: false
```

4.6 Stage Dependencies

Explicitly define execution order (rarely needed).

```yaml
stages:
 stage_a:
  cmd: python a.py
  outs:
   - output_a.csv

 stage_b:
  cmd: python b.py
  deps:
   - output_a.csv     # Implicit dependency on stage_a
  outs:
   - output_b.csv

 stage_c:
  cmd: python c.py
  deps:
   - output_b.csv     # Implicit dependency on stage_b
  outs:
   - output_c.csv
```

Note: DVC automatically determines execution order from file dependencies. Explicit stage dependencies are rarely needed.

5 Complete ML Pipeline Example

5.1 Project Structure

```
ml_project/
|-- dvc.yaml
|-- params.yaml
|-- .dvc/
|   |-- config
|   `-- cache/
|-- data/
|   |-- raw/
|   |-- processed/
|   `-- features/
|-- models/
|-- metrics/
|-- plots/
|-- src/
|   |-- data_collection.py
|   |-- preprocessing.py
|   |-- feature_engineering.py
|   |-- train.py
|   |-- evaluate.py
|   `-- models/
|       `-- architectures.py
|-- notebooks/
|-- requirements.txt
`-- README.md
```

5.2 Complete dvc.yaml

```yaml
vars:
 - data_dir: data
 - models_dir: models
 - metrics_dir: metrics
 - plots_dir: plots
 - src_dir: src

stages:
 # ==========================================
 # Stage 1: Data Collection
 # ==========================================
 collect_data:
  cmd: python ${src_dir}/data_collection.py
  deps:
   - ${src_dir}/data_collection.py
  params:
   - data_collection.source_url
   - data_collection.date_range
   - data_collection.api_version
  outs:
   - ${data_dir}/raw/dataset.csv
   - ${data_dir}/raw/metadata.json

 # ==========================================
 # Stage 2: Data Preprocessing
 # ==========================================
 preprocess:
  cmd: python ${src_dir}/preprocessing.py
  deps:
   - ${data_dir}/raw/dataset.csv
   - ${src_dir}/preprocessing.py
  params:
   - preprocessing.train_split
   - preprocessing.test_split
   - preprocessing.validation_split
   - preprocessing.random_seed
   - preprocessing.handle_missing
   - preprocessing.remove_outliers
  outs:
   - ${data_dir}/processed/train.csv
   - ${data_dir}/processed/test.csv
   - ${data_dir}/processed/validation.csv
   - ${data_dir}/processed/stats.json

 # ==========================================
 # Stage 3: Feature Engineering
 # ==========================================
 feature_engineering:
  cmd: python ${src_dir}/feature_engineering.py
  deps:
   - ${data_dir}/processed/train.csv
   - ${data_dir}/processed/test.csv
   - ${data_dir}/processed/validation.csv
   - ${src_dir}/feature_engineering.py
  params:
   - features.numerical_features
   - features.categorical_features
   - features.text_features
   - features.scaling_method
   - features.encoding_method
  outs:
   - ${data_dir}/features/X_train.npy
   - ${data_dir}/features/X_test.npy
   - ${data_dir}/features/X_val.npy
   - ${data_dir}/features/y_train.npy
   - ${data_dir}/features/y_test.npy
   - ${data_dir}/features/y_val.npy
   - ${models_dir}/preprocessors/scaler.pkl
   - ${models_dir}/preprocessors/encoder.pkl

 # ==========================================
 # Stage 4: Model Training
 # ==========================================
 train:
  cmd: python ${src_dir}/train.py
  deps:
   - ${data_dir}/features/X_train.npy
   - ${data_dir}/features/X_val.npy
   - ${data_dir}/features/y_train.npy
   - ${data_dir}/features/y_val.npy
   - ${src_dir}/train.py
   - ${src_dir}/models/architectures.py
  params:
   - training.model_type
   - training.epochs
   - training.batch_size
   - training.learning_rate
   - training.optimizer
   - training.loss_function
   - training.early_stopping
   - training.random_seed
  outs:
   - ${models_dir}/trained/model.pkl
   - ${models_dir}/trained/weights.h5
   - ${models_dir}/trained/history.json
  plots:
   - ${plots_dir}/training_loss.csv:
     cache: false
     x: epoch
     y: [loss, val_loss]
     title: "Training vs Validation Loss"
   - ${plots_dir}/training_accuracy.csv:
     cache: false
     x: epoch
     y: [accuracy, val_accuracy]

 # ===========================================
 # Stage 5: Model Evaluation
 # ===========================================
 evaluate:
  cmd: python ${src_dir}/evaluate.py
  deps:
   - ${data_dir}/features/X_test.npy
   - ${data_dir}/features/y_test.npy
   - ${models_dir}/trained/model.pkl
   - ${models_dir}/preprocessors/scaler.pkl
   - ${src_dir}/evaluate.py
  params:
   - evaluation.metrics
   - evaluation.threshold
  outs:
   - predictions/test_predictions.csv
  metrics:
   - ${metrics_dir}/test_scores.json:
     cache: false
  plots:
   - ${plots_dir}/confusion_matrix.csv:
     cache: false
     template: confusion
   - ${plots_dir}/roc_curve.json:
     cache: false
     x: fpr
     y: tpr
     title: "ROC Curve"
   - ${plots_dir}/precision_recall.json:
     cache: false
     x: recall
     y: precision
```

5.3 Complete params.yaml

```yaml
# ==========================================
# Data Collection Parameters
# ==========================================
data_collection:
 source_url: "https://api.example.com/v2/dataset"
 date_range:
  start: "2024-01-01"
  end: "2024-12-31"
 api_version: "v2.0"
 timeout: 30

# ==========================================
# Preprocessing Parameters
# ==========================================
preprocessing:
 train_split: 0.70
 test_split: 0.20
 validation_split: 0.10
 random_seed: 42
 handle_missing: "median"
 remove_outliers: true
 outlier_method: "iqr"
 outlier_threshold: 1.5

# ==========================================
# Feature Engineering
# ==========================================
features:
 numerical_features:
  - age
  - income
  - credit_score
  - balance

 categorical_features:
  - gender
  - occupation
  - education

 text_features:
  - description

 scaling_method: "standard"
 encoding_method: "onehot"

# ==========================================
# Training Parameters
# ==========================================
training:
 model_type: "random_forest"
 epochs: 100
 batch_size: 32
 learning_rate: 0.001
 optimizer: "adam"
 loss_function: "binary_crossentropy"
 early_stopping:
    enabled: true
    patience: 10
    min_delta: 0.001
  random_seed: 42

# ==========================================
# Evaluation Parameters
# ==========================================
evaluation:
  metrics:
    - accuracy
    - precision
    - recall
    - f1_score
    - roc_auc
  threshold: 0.5
```

6 Essential DVC Commands

6.1 Initialization and Setup

```
# Initialize DVC in project
dvc init

# Configure S3 remote
dvc remote add -d storage s3://mybucket/dvcstore

# Configure GCS remote
dvc remote add -d storage gs://mybucket/dvcstore

# List remotes
dvc remote list

# Modify remote URL
dvc remote modify storage url s3://newbucket/path
```

6.2 Pipeline Operations

```
# Run entire pipeline
dvc repro

# Run specific stage
dvc repro train

# Force re-run all stages
dvc repro -f

# Force re-run specific stage
dvc repro -f preprocess

# Show pipeline DAG
dvc dag

# Check pipeline status
dvc status

# Detailed status
dvc status -v
```

6.3 Data Management

```
# Track large file
dvc add data/dataset.csv

# Track directory
dvc add data/images/
```

```
# Push data to remote
dvc push

# Pull data from remote
dvc pull

# Fetch to cache only
dvc fetch

# Update workspace from cache
dvc checkout
```

6.4 Metrics and Plots

```
# Show metrics
dvc metrics show

# Compare metrics with branch
dvc metrics diff main

# Show plots
dvc plots show

# Compare plots
dvc plots diff main experiment

# Generate HTML report
dvc plots show --html
```

6.5 Experiments

```
# Run experiment
dvc exp run

# Run with parameter override
dvc exp run -S train.lr=0.01

# Queue experiments
dvc exp run --queue -S train.lr=0.001
dvc exp run --queue -S train.lr=0.01
dvc queue start

# List experiments
dvc exp show

# Compare experiments
dvc exp diff
```

```
# Apply experiment
dvc exp apply exp-12345
```

7 Best Practices

7.1 Pipeline Design

Good Practices:

- Create stages for logical workflow steps
- Keep stages focused (single responsibility)
- Use descriptive stage names (`preprocess_data`, not `stage1`)
- Track all dependencies accurately
- Organize outputs in structured directories
- Use `params.yaml` for all configurable values

Avoid:

- Overly granular stages (creates overhead)
- Combining unrelated operations in one stage
- Missing dependencies (breaks reproducibility)
- Hardcoding parameters in commands
- Caching log files and metrics

7.2 Version Control Integration

7.2.1 Track with Git

- `dvc.yaml` - Pipeline definition
- `dvc.lock` - Lock file
- `params.yaml` - Parameters
- `*.dvc` - File pointers
- `.dvc/config` - Configuration
- `src/` - Source code

7.2.2 Track with DVC

- `data/` - Datasets
- `models/` - Trained models
- Large files (>1MB)

7.3 Parameter Management

```yaml
# Good: Hierarchical structure
preprocessing:
 train_split: 0.7
 random_seed: 42

training:
 epochs: 100
 learning_rate: 0.001

# Bad: Flat structure
train_split: 0.7
epochs: 100
learning_rate: 0.001
```

8 Troubleshooting

8.1 Pipeline Won't Run

Problem: `dvc repro` doesn't execute stages

Possible Causes:

- Pipeline already up-to-date (no changes detected)
- Syntax errors in `dvc.yaml`
- Missing dependencies
- Circular dependencies
- Frozen stages

```
# Check pipeline status
dvc status
dvc status -v

# Validate pipeline structure
dvc dag

# Force re-run entire pipeline
dvc repro -f

# Force re-run specific stage
dvc repro -f stage_name

# Check for syntax errors
cat dvc.yaml | python -m yaml

# Dry run (show what would execute)
dvc repro --dry
```

8.2 Stage Always Re-runs

Problem: Stage executes every time despite no changes

Possible Causes:

- Missing dependencies in `deps` field
- Files being modified by external processes
- Non-deterministic code (random operations without seed)
- Timestamp issues
- Command generates different output each time

Solutions:

```
# Check what DVC detects as changed
dvc status -v

# Verify file checksums
dvc status --show-json

# Set random seeds in your code
# Python example:
import random
import numpy as np
random.seed(42)
np.random.seed(42)
```

Ensure Reproducibility:

- List ALL dependencies (data, code, configs)
- Set random seeds for all random operations
- Avoid timestamp-based operations
- Use deterministic algorithms

8.3 Missing Data Files

Problem: Data files not found after cloning repository

```
# Pull all data from remote
dvc pull

# Pull specific file
dvc pull data/dataset.csv.dvc

# Check remote configuration
dvc remote list
dvc remote list --show-origin

# Test connectivity to remote
dvc status --cloud

# Check if files are in cache
ls -la .dvc/cache/

# Fetch to cache without checking out
dvc fetch
dvc checkout
```

8.4 Push/Pull Failures

Problem: Cannot push or pull data to/from remote

Common Issues:

- Incorrect remote URL
- Missing credentials
- Permission issues
- Network connectivity
- Insufficient storage quota

```
# Check remote configuration
dvc remote list
dvc config remote.storage.url

# Verbose output for debugging
dvc push -v
dvc pull -v

# Check cloud status
dvc status --cloud

# For AWS S3 - verify credentials
aws configure list
aws s3 ls s3://your-bucket/

# For Google Cloud - verify authentication
gcloud auth list
gsutil ls gs://your-bucket/

# Test with small file first
dvc add test.txt
dvc push test.txt.dvc
```

8.5 Cache Corruption

Problem: DVC cache becomes corrupted or inconsistent

```
# Check cache integrity
ls -la .dvc/cache/

# Remove corrupted cache
rm -rf .dvc/cache

# Re-download from remote
dvc fetch
dvc checkout

# Or re-run pipeline
dvc repro -f

# Verify data integrity
dvc status
```

8.6 Disk Space Issues

Problem: DVC cache consuming too much disk space

```
# Check cache size
du -sh .dvc/cache
dvc cache dir

# Show what would be removed (dry run)
dvc gc --dry

# Remove files not in current workspace
dvc gc --workspace

# Keep workspace, all branches, and all tags
dvc gc --workspace --all-branches --all-tags

# Remove all except workspace
dvc gc --workspace --force

# Clean cloud cache
dvc gc --cloud

# Check after cleanup
du -sh .dvc/cache
```

Warning: Be careful with `dvc gc`. Always use `--dry` first to see what would be removed. Keep backups of important data.

8.7 Lock File Conflicts

Problem: Git merge conflict in `dvc.lock`

Solution 1: Accept One Version

```
# Accept your version
git checkout --ours dvc.lock

# Or accept their version
git checkout --theirs dvc.lock

# Regenerate lock file
dvc repro

# Commit resolution
git add dvc.lock
git commit -m "Resolve dvc.lock conflict"
```

Solution 2: Regenerate

```
# Remove conflicted lock file
rm dvc.lock

# Regenerate from pipeline
dvc repro

# Commit new lock file
git add dvc.lock
git commit -m "Regenerate dvc.lock"
```

8.8 Permission Issues

Problem: Permission denied errors

```
# Fix local cache permissions
chmod -R u+w .dvc/cache

# Fix workspace file permissions
chmod -R u+w data/ models/

# For remote storage (S3)
# Check IAM permissions in AWS Console

# For SSH remotes
ssh user@host "chmod -R u+w /path/to/dvc/storage"

# Check file ownership
ls -la .dvc/cache
ls -la data/
```

8.9 Performance Issues

8.9.1 Slow Pipeline Execution

Optimizations:

- Enable parallel stage execution (if independent)
- Use faster remote storage region
- Optimize data I/O operations in code
- Use data sampling for development
- Profile code for bottlenecks
- Consider stage granularity
- Use SSD for cache directory

8.9.2 Slow Push/Pull Operations

```
# Use faster remote region
dvc remote modify storage region us-west-2

# Enable compression
dvc remote modify storage --local compression gzip

# Increase transfer workers
dvc remote modify storage --local upload_max_workers 20
dvc remote modify storage --local download_max_workers 20

# Use jobs parameter
dvc push --jobs 8
dvc pull --jobs 8
```

8.10 Import/Module Errors

Problem: Python modules not found during stage execution

```
# Verify Python environment
which python
python --version

# Check installed packages
pip list

# Install requirements
pip install -r requirements.txt

# Check PYTHONPATH
echo $PYTHONPATH

# Add to PYTHONPATH in stage command
stages:
  train:
    cmd: PYTHONPATH=src:$PYTHONPATH python src/train.py
```

9 Advanced Topics

9.1 Remote Storage Configuration

9.1.1 AWS S3

```
# Add S3 remote
dvc remote add -d storage s3://mybucket/dvcstore

# Configure region
dvc remote modify storage region us-west-2

# Use AWS profile
dvc remote modify storage profile myprofile

# Use specific credentials (not recommended)
dvc remote modify storage access_key_id YOUR_KEY
dvc remote modify storage secret_access_key YOUR_SECRET

# Enable server-side encryption
dvc remote modify storage sse AES256

# Use custom endpoint (MinIO, etc.)
dvc remote modify storage endpointurl http://localhost:9000

# Enable versioning
dvc remote modify storage version_aware true
```

9.1.2 Google Cloud Storage

```
# Add GCS remote
dvc remote add -d storage gs://mybucket/dvcstore

# Configure project
dvc remote modify storage projectname myproject

# Use service account
dvc remote modify storage credentialpath \
    /path/to/credentials.json

# Set default credentials
export GOOGLE_APPLICATION_CREDENTIALS=/path/to/key.json
```

9.1.3 Azure Blob Storage

```
# Add Azure remote
dvc remote add -d storage azure://mycontainer/path

# Use connection string
```

```
dvc remote modify storage connection_string \
    "DefaultEndpointsProtocol=https;..."

# Or use account name and key
dvc remote modify storage account_name myaccount
dvc remote modify storage account_key mykey

# Use SAS token
dvc remote modify storage sas_token "?sv=2019..."
```

9.1.4 SSH Remote

```
# Add SSH remote
dvc remote add -d storage \
    ssh://user@example.com/path/to/dvc

# Use SSH key
dvc remote modify storage keyfile ~/.ssh/id_rsa

# Use password (not recommended)
dvc remote modify storage password mypassword

# Specify port
dvc remote modify storage port 2222

# Use SSH config
dvc remote modify storage ssh_config ~/.ssh/config
```

9.2 CI/CD Integration

9.2.1 GitHub Actions

```
# .github/workflows/dvc-pipeline.yml
name: DVC Pipeline

on:
 push:
  branches: [main]
 pull_request:
  branches: [main]

jobs:
 run-pipeline:
  runs-on: ubuntu-latest

  steps:
   - name: Checkout code
    uses: actions/checkout@v3

   - name: Setup Python
    uses: actions/setup-python@v4
    with:
     python-version: '3.10'

   - name: Install dependencies
    run: |
     pip install dvc[s3]
     pip install -r requirements.txt

   - name: Configure AWS credentials
    uses: aws-actions/configure-aws-credentials@v2
    with:
     aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
     aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
     aws-region: us-west-2

   - name: Pull DVC data
    run: dvc pull

   - name: Run pipeline
    run: dvc repro

   - name: Show metrics
    run: dvc metrics show

   - name: Push results
    if: github.ref == 'refs/heads/main'
    run: dvc push

   - name: Create CML report
    run: |
     echo "## Metrics" >> report.md
     dvc metrics show --md >> report.md
     cml comment create report.md
```

9.2.2 GitLab CI

```
# .gitlab-ci.yml
image: python:3.10

stages:
 - setup
 - pipeline
 - report

variables:
 DVC_REMOTE: s3://mybucket/dvcstore

before_script:
 - pip install dvc[s3]
 - pip install -r requirements.txt

setup_dvc:
 stage: setup
 script:
  - dvc remote modify storage access_key_id $AWS_ACCESS_KEY_ID
  - dvc remote modify storage secret_access_key $AWS_SECRET_ACCESS_KEY
  - dvc pull
 cache:
    paths:
      - .dvc/cache

run_pipeline:
  stage: pipeline
  script:
    - dvc repro
    - dvc push
  artifacts:
    paths:
      - metrics/
      - plots/
    expire_in: 1 week

generate_report:
  stage: report
  script:
    - dvc metrics show
    - dvc plots show
  artifacts:
    paths:
      - metrics/
      - plots/
```

9.3 Custom Metrics and Plots

9.3.1 Custom Metrics Structure

```json
{
  "train": {
    "accuracy": 0.9542,
    "loss": 0.1234,
    "precision": 0.9321,
    "recall": 0.9456
  },
  "validation": {
    "accuracy": 0.9312,
    "loss": 0.1567,
    "precision": 0.9123,
    "recall": 0.9287
  },
  "test": {
    "accuracy": 0.9278,
    "loss": 0.1689,
    "precision": 0.9056,
    "recall": 0.9201
  },
  "model_info": {
    "parameters": 1250000,
    "training_time": 3600,
    "epochs_trained": 50
  }
}
```

9.3.2 Custom Plot Templates

```yaml
stages:
 evaluate:
  plots:
   - plots/custom_metrics.csv:
     template: linear
     x: iteration
     y:
      plots/custom_metrics.csv: [metric_a, metric_b]
     title: "Custom Metrics Comparison"
     x_label: "Training Iteration"
     y_label: "Metric Value"

   - plots/scatter_plot.csv:
     template: scatter
     x: predicted
     y: actual
     title: "Predicted vs Actual"

   - plots/smooth_curve.csv:
     template: smooth
     x: epoch
     y: loss
     title: "Smoothed Loss Curve"
```

9.4 Pipeline Modularization

9.4.1 Splitting Large Pipelines

```
project/
|-- dvc.yaml
|-- pipelines/
|   |-- data_pipeline.yaml
|   |-- training_pipeline.yaml
|   ‘-- evaluation_pipeline.yaml
|-- params/
|   |-- data_params.yaml
|   |-- training_params.yaml
|   ‘-- evaluation_params.yaml
```

```yaml
# pipelines/data_pipeline.yaml
stages:
 collect:
  cmd: python src/collect.py
  deps:
   - src/collect.py
  outs:
   - data/raw/dataset.csv

 preprocess:
  cmd: python src/preprocess.py
  deps:
   - data/raw/dataset.csv
   - src/preprocess.py
  outs:
   - data/processed/train.csv
   - data/processed/test.csv
```

```
# dvc.yaml (main pipeline)
stages:
 data_pipeline:
  cmd: dvc repro pipelines/data_pipeline.yaml

 training_pipeline:
  cmd: dvc repro pipelines/training_pipeline.yaml
  deps:
   - data/processed/train.csv

 evaluation_pipeline:
  cmd: dvc repro pipelines/evaluation_pipeline.yaml
  deps:
   - models/trained/model.pkl
```

9.5 Experiment Management

9.5.1 Running Multiple Experiments

```
# Queue multiple experiments
dvc exp run --queue -S train.lr=0.001 -S train.epochs=50
dvc exp run --queue -S train.lr=0.01 -S train.epochs=50
dvc exp run --queue -S train.lr=0.1 -S train.epochs=50
dvc exp run --queue -S train.lr=0.001 -S train.epochs=100

# Start queue execution
dvc queue start

# Run experiments in parallel (4 workers)
dvc queue start --jobs 4

# Check queue status
dvc queue status

# Stop queue
dvc queue stop

# Remove experiments from queue
dvc queue remove exp-12345
```

9.5.2 Comparing Experiments

```
# List all experiments
dvc exp show

# Show only changed parameters and metrics
dvc exp show --only-changed
```

```
# Show experiments in table format
dvc exp show --no-pager

# Compare specific experiments
dvc exp diff exp-abc123 exp-def456

# Compare with specific branch
dvc exp diff main

# Show differences in specific metrics
dvc exp show --include-params train.lr \
    --include-metrics accuracy,f1_score
```

9.5.3 Managing Experiments

```
# Apply experiment to workspace
dvc exp apply exp-abc123

# Create branch from experiment
dvc exp branch exp-abc123 feature/best-model

# Push experiment to remote
dvc exp push origin exp-abc123

# Pull experiments from remote
dvc exp pull origin

# Remove experiment
dvc exp remove exp-abc123

# Remove all experiments
dvc exp remove --all

# Remove experiments in queue
dvc exp remove --queue
```

10 Quick Reference Guide

10.1 dvc.yaml Components Summary

Component	Description
`stages`	Define individual pipeline stages
`cmd`	Shell command to execute
`wdir`	Working directory for execution
`deps`	File/directory dependencies
`outs`	Output files/directories
`params`	Parameter references from `params.yaml`
`metrics`	Evaluation metric files (JSON/YAML/CSV)
`plots`	Plot data files for visualization
`frozen`	Prevent automatic execution
`always_changed`	Force re-execution every time
`foreach`	Loop over items
`vars`	Define reusable variables

10.2 Command Cheat Sheet

Initialization & Setup

```
dvc init                        # Initialize DVC
dvc remote add -d name url      # Add default remote
dvc remote list                 # List remotes
```

Pipeline Operations

```
dvc repro                       # Run pipeline
dvc repro -f                    # Force re-run
dvc repro stage_name            # Run specific stage
dvc dag                         # Show pipeline DAG
dvc status                      # Check status
```

Data Management

```
dvc add file                    # Track file
dvc push                        # Push to remote
dvc pull                        # Pull from remote
dvc fetch                       # Fetch to cache
dvc checkout                    # Update workspace
```

Metrics & Plots

```
dvc metrics show                # Show metrics
dvc metrics diff                # Compare metrics
dvc plots show                  # Show plots
dvc plots diff                  # Compare plots
```

Experiments

```
dvc exp run                    # Run experiment
dvc exp show                   # List experiments
dvc exp diff                   # Compare experiments
dvc exp apply exp-id           # Apply experiment
dvc exp branch exp-id name     # Create branch
```

10.3 Common Workflows

10.3.1 Start New DVC Project

```
# Initialize repositories
git init
dvc init

# Configure remote storage
dvc remote add -d storage s3://mybucket/dvcstore

# Commit configuration
git add .dvc .gitignore
git commit -m "Initialize DVC"

# Track large dataset
dvc add data/dataset.csv
git add data/dataset.csv.dvc data/.gitignore
git commit -m "Track dataset"

# Push data to remote
dvc push
```

10.3.2 Clone DVC Project

```
# Clone repository
git clone https://github.com/user/repo.git
cd repo

# Pull data from remote
dvc pull

# Verify pipeline
dvc dag
dvc status

# Run pipeline
dvc repro
```

10.3.3 Run Experiment with Different Parameters

```
# Create feature branch
git checkout -b experiment/new-model

# Modify parameters
vim params.yaml

# Run pipeline
dvc repro

# Compare with main branch
dvc metrics diff main
dvc plots diff main

# If successful, commit and merge
git add params.yaml dvc.lock
git commit -m "Experiment: improved model"
dvc push
git push origin experiment/new-model
```

10.3.4 Hyperparameter Tuning

```
# Queue multiple experiments
for lr in 0.001 0.01 0.1; do
  for bs in 16 32 64; do
    dvc exp run --queue \
      -S train.learning_rate=$lr \
      -S train.batch_size=$bs \
      --name "lr${lr}_bs${bs}"
  done
done

# Run all queued experiments
dvc queue start --jobs 4

# Compare results
dvc exp show --only-changed

# Apply best experiment
dvc exp apply exp-best-id
git add params.yaml dvc.lock
git commit -m "Apply best hyperparameters"
```

11 Glossary

Cache	Local storage directory (`.dvc/cache`) where DVC stores actual file contents, indexed by hash.
Checkpoint	Special type of output that can be saved incrementally during training, useful for long-running experiments and allowing resumption.
DAG	Directed Acyclic Graph representing the pipeline's stage dependencies and execution order. Ensures no circular dependencies.
Dependency	File or directory that a stage depends on. Changes to dependencies trigger stage re-execution during `dvc repro`.
DVC File	Files with `.dvc` extension containing metadata and pointers to actual data stored in cache or remote storage.
dvc.lock	Lock file containing exact checksums and dependency versions, ensuring reproducibility across different environments.
dvc.yaml	Central pipeline definition file describing stages, dependencies, outputs, parameters, metrics, and plots.
Experiment	A variation of pipeline execution with different parameters, code, or data, tracked by DVC for comparison.
Frozen Stage	Stage marked with `frozen: true` that won't execute automatically during `dvc repro`.
Metric	Numeric evaluation measure (accuracy, loss, F1) stored in JSON/YAML/CSV format for performance tracking.
Output	File or directory produced by a stage, tracked and cached by DVC for efficiency and reproducibility.
Parameter	Configuration value stored in `params.yaml`, tracked for experiment comparison and version control.
Pipeline	Series of connected stages forming a complete ML workflow, defined in `dvc.yaml`.
Plot	Data file (CSV/JSON) used for generating visualizations like learning curves, ROC curves, or confusion matrices.
Remote Storage	Cloud or network storage location (S3, GCS, Azure, SSH) for sharing DVC-tracked data with team members.
Repro	Short for "reproduce" - the `dvc repro` command that runs pipeline stages based on their dependencies.
Stage	Single step in DVC pipeline with defined command, dependencies, outputs, and optional parameters/metrics.
Workspace	Project's working directory containing checked-out files, source code, and pipeline configuration.

12 Conclusion

12.1 Key Takeaways

DVC Provides:

- **Version Control:** Git-like interface for data and models
- **Reproducibility:** Guaranteed consistent results across runs
- **Experimentation:** Track and compare parameter variations
- **Collaboration:** Share data and pipelines with teams
- **Scalability:** Handle datasets of any size
- **Flexibility:** Works with any ML framework and cloud provider

12.2 Getting Started Checklist

1. **Initialize:** Set up Git and DVC in your project
2. **Configure Remote:** Add cloud storage for data sharing
3. **Track Data:** Use `dvc add` for large files
4. **Define Pipeline:** Create `dvc.yaml` with stages
5. **Set Parameters:** Configure `params.yaml`
6. **Run Pipeline:** Execute with `dvc repro`
7. **Track Metrics:** Monitor performance over time
8. **Version Control:** Commit `dvc.yaml` and `dvc.lock` to Git
9. **Share:** Push data with `dvc push`
10. **Collaborate:** Pull team changes with `git pull` and `dvc pull`

12.3 Best Practices Recap

Remember:

- Track ALL dependencies (data, code, configs)
- Use `params.yaml` for all configurable values
- Never cache metrics or plots
- Set random seeds for reproducibility
- Commit `dvc.yaml` and `dvc.lock` to Git
- Push data regularly to remote storage
- Document your pipeline for team understanding
- Use descriptive stage and parameter names

12.4 Next Steps

1. **Practice**: Start with a small project
2. **Experiment**: Try different pipeline structures
3. **Integrate**: Add CI/CD automation
4. **Explore**: Use experiment tracking features
5. **Optimize**: Fine-tune performance and storage
6. **Share**: Collaborate with your team

12.5 Additional Resources

- **Documentation**: `https://dvc.org/doc`
- **Tutorial**: `https://dvc.org/doc/start`
- **GitHub**: `https://github.com/iterative/dvc`
- **Community**: `https://discuss.dvc.org`
- **Blog**: `https://dvc.org/blog`
- **YouTube**: `https://www.youtube.com/c/DVCorg`
- **Examples**: `https://github.com/iterative/example-get-started`

Thank You!

End of DVC Pipeline Configuration Guide

Created by Sujil S — sujil9480@gmail.com