# Comprehensive Guide to Coding Productivity

Reference Manual for Machine Learning Operations

March 6, 2025

**By**
Arshia Ilaty

# Contents

# 1 Introduction

This comprehensive guide is designed to enhance productivity for machine learning researchers and developers working with remote servers, GPU infrastructure, and distributed training systems. The document organizes essential commands, workflows, and best practices for various operational tasks encountered during machine learning research and development.

# 2 Project Structure

A well-organized project structure enhances collaboration and productivity. Below is the recommended structure for machine learning projects:

```
project-root/
          data/                # Datasets directory
                raw/             # Raw unprocessed data
                processed/       # Processed data ready for modeling
                external/        # Data from external sources
          models/              # Saved models and checkpoints
          notebooks/           # Jupyter notebooks for exploration
          src/                 # Source code
                __init__.py
                data/            # Data processing scripts
                features/        # Feature engineering scripts
                models/          # Model implementation
                utils/           # Utility functions
          configs/             # Configuration files
          experiments/         # Experiment logs and results
          scripts/             # Automation scripts
          tests/               # Test files
          requirements.txt     # Dependencies
          setup.py             # Package setup
          README.md            # Project documentation
```

Listing 1: Basic Project Structure

# 3 Remote Server Operations

## 3.1 SSH Connection and Authentication

Secure shell (SSH) connection is essential for remote server access.

**SSH Connection Commands**

```
# Basic SSH connection
ssh username@server-address

# Example with specific port
ssh -p PORT_NUMBER username@server-address

# Using SSH key for authentication
ssh -i /path/to/private_key username@server-address
```

## 3.2 VPN Connection

VPN connection for secure access to internal resources.

VPN Connection (MacOS)

```
# Control GlobalProtect VPN via AppleScript
echo 'tell application "System Events"
    tell process "GlobalProtect"
        click menu bar item 1 of menu bar 2
    end tell
end tell' > gp-control.scpt

# Execute the script
osascript gp-control.scpt
```

# 4 Command Extraction and History Management

## 4.1 Command History Basics

Command history tracking is essential for reproducibility in machine learning projects. Understanding different ways to extract and analyze your command history can save significant time when revisiting projects or sharing workflows with colleagues.

Basic Command History

```
# View command history
history

# Export all command history
history > command_history.txt

# Search command history
history | grep "keyword"

# Export the last N commands
history N > recent_commands.txt
```

## 4.2 Advanced Command History Filtering

For more complex projects, filtering command history by date, pattern, or context can be invaluable.

Advanced Command History Filtering

```
# Filter commands from the last 7 days
history | awk -v date="$(date -d '7 days ago' '+%F')" '$2 >= date' >
    recent_commands.txt

# Extract commands for a specific project
history | grep "project_name" > project_commands.txt

# Find all Python script executions
history | grep "python" > python_commands.txt

# Extract GPU-related commands
history | grep -E 'nvidia|cuda|gpu' > gpu_commands.txt
```

## 4.3 Project-Specific Command Extraction

For machine learning projects with multiple components, extracting commands by project directory can help with documentation.

Project-Specific Command Extraction

```
# Extract commands for a specific project path
cat ~/.bash_history | grep -i "project_name"

# Extract all conda environment activations
cat ~/.bash_history | grep -i "conda activate"

# Find all commands run in a specific directory
cat ~/.bash_history | grep -i "cd directory_name" -A 5

# Extract all training commands
cat ~/.bash_history | grep -i "python.*train"
```

# 5 File Operations

## 5.1 File Transfer Between Local and Remote

Efficient file transfer commands for moving data between local and remote systems.

File Transfer with SCP

```
# From local to remote server
scp /local/path/file.csv username@server:/remote/path/

# From remote server to local
scp username@server:/remote/path/file.csv /local/path/

# For directories (recursive)
scp -r /local/directory username@server:/remote/path/

# With specific port
scp -P PORT_NUMBER /local/path/file.csv username@server:/remote/path/

# With SSH key
scp -i /path/to/key.pem /local/path/file.csv username@server:/remote/path/
```

## 5.2 Basic File Management

Essential file operations for organizing your workspace.

File Management Commands

```
# Moving files
mv /path/to/source /path/to/destination

# Copying files
cp source_file destination_file

# Copying directories (recursive)
cp -r source_directory destination_directory

# Removing files
rm filename

# Removing directories (recursive)
rm -r directory_name

# Creating backup of a directory
cp -r project/ project-backup/

# Finding large files
find /path/to/search -type f -size +100M | sort -k 5 -nr
```

# 6 GPU and Computing Resources

## 6.1 Monitoring GPU Usage

Commands for tracking GPU resource utilization.

GPU Monitoring

```
# Monitor GPU usage in real-time
watch -n 1 nvidia-smi

# Get detailed GPU information
nvidia-smi -q

# Monitor specific metrics
nvidia-smi --query-gpu=timestamp,name,utilization.gpu,utilization.memory,memory.
    used,memory.total --format=csv -l 1
```

## 6.2 CUDA Management

Optimizing CUDA for better GPU usage.

CUDA Environment Setup

```
# Avoid memory fragmentation in PyTorch
export PYTORCH_CUDA_ALLOC_CONF="expandable_segments:True"

# Explicitly select GPU device
export CUDA_VISIBLE_DEVICES="0"
export CUDA_DEVICE_ORDER="PCI_BUS_ID"

# Check CUDA version
nvcc --version

# List available CUDA devices
python -c "import torch; print(torch.cuda.device_count())"
```

PyTorch CUDA Configuration Code

```python
import os
import torch

# Add this to avoid memory fragmentation
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"

# Add explicit device selection for MIG environment
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"

# Check CUDA availability
print(f"CUDA available: {torch.cuda.is_available()}")
print(f"CUDA device count: {torch.cuda.device_count()}")
if torch.cuda.is_available():
    print(f"Current CUDA device: {torch.cuda.current_device()}")
    print(f"Device name: {torch.cuda.get_device_name()}")
```

## 6.3 Creating Persistent CUDA Environments

Setting up persistent conda environments for CUDA development in shared platforms like JupyterHub.

Persistent CUDA Environment Setup

```bash
#!/bin/bash
# Create a persistent CUDA environment
# Usage: bash persistent_setup_cuda_env.sh [env_name] [python_version]

# Default values
ENV_NAME=${1:-"cuda_env"}
PYTHON_VERSION=${2:-"3.10"}
PERSISTENT_DIR="/home/username/persistent_envs"  # Use a persistent location

# Create persistent directory
mkdir -p $PERSISTENT_DIR

echo "=== Setting up persistent CUDA environment ==="
echo "Environment name: $ENV_NAME"
echo "Python version: $PYTHON_VERSION"
echo "Location: $PERSISTENT_DIR/$ENV_NAME"

# Initialize conda
CONDA_BASE=$(conda info --base)
source "$CONDA_BASE/etc/profile.d/conda.sh"

# Create environment in persistent location
conda create -y -p "$PERSISTENT_DIR/$ENV_NAME" python="$PYTHON_VERSION"
conda activate "$PERSISTENT_DIR/$ENV_NAME"

# Install PyTorch with CUDA support
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/
    whl/cu124

# Install additional packages
conda install -y -c conda-forge numpy scipy pandas matplotlib jupyterlab
pip install transformers accelerate

# Create activation script
ACTIVATE_SCRIPT="/home/username/activate_$ENV_NAME.sh"
cat > "$ACTIVATE_SCRIPT" << EOF
#!/bin/bash
# Initialize conda
source "\$(conda info --base)/etc/profile.d/conda.sh"

# Activate by PATH not by NAME
conda activate "$PERSISTENT_DIR/$ENV_NAME"

# Check CUDA status
nvidia-smi
python -c "import torch; print('CUDA available:', torch.cuda.is_available())"
EOF

chmod +x "$ACTIVATE_SCRIPT"
echo "Created activation script: $ACTIVATE_SCRIPT"
```

## 6.4 Why Use Path-Based Environments?

In shared environments like JupyterHub, conda environments are often automatically deleted after periods of inactivity. This behavior affects environments activated by name but not those activated by path.

Named vs. Path-Based Conda Environments

| Named Environment | Path-Based Environment |
|---|---|
| `conda create -n env_name` | `conda create -p /path/to/env` |
| `conda activate env_name` | `conda activate /path/to/env` |
| Stored in Conda's central location | Stored in custom location |
| Subject to auto-cleanup | Persists through cleanup cycles |
| Easier syntax | More verbose but more persistent |

Key advantages of path-based environments:

- Survive automatic cleanup in shared environments
- Can be placed in backed-up or shared directories
- Facilitate team collaboration with identical environments
- Provide explicit location awareness

## 6.5 Clearing CUDA Cache

Cleaning up GPU memory for better resource utilization.

CUDA Cache Management

```python
# Clear PyTorch CUDA cache in Python
import torch
torch.cuda.empty_cache()

# Memory management in PyTorch code
def clear_gpu_memory():
    import gc
    import torch
    gc.collect()
    torch.cuda.empty_cache()

    # Print memory stats for verification
    print(f"Memory allocated: {torch.cuda.memory_allocated() / 1e9:.2f} GB")
    print(f"Memory reserved: {torch.cuda.memory_reserved() / 1e9:.2f} GB")
```

# 7 Disk Space Management and Cleanup

## 7.1 Identifying Disk Usage

Commands to identify what's consuming disk space on your machine learning environment.

Disk Usage Analysis

```bash
# Check overall disk usage
df -h

# Check size of specific directory
du -sh /path/to/directory

# Find the largest directories (depth=1)
du -h --max-depth=1 /home/username | sort -hr

# Find large files (>100MB)
find /home/username -type f -size +100M | xargs du -h | sort -hr
```

## 7.2 Cache Management

ML environments often accumulate large caches that can be safely cleaned.

Cache Cleanup

```
# Check pip cache size
du -sh ~/.cache/pip/

# Clear pip cache
pip cache purge

# Check conda cache
du -sh ~/.conda/pkgs/

# Clean conda cache
conda clean -a -y

# Check Hugging Face cache (often very large)
du -sh ~/.cache/huggingface/

# Selectively remove Hugging Face models
rm -rf ~/.cache/huggingface/hub/models--specific-model-name
```

## 7.3 Managing Large Model Caches

When working with large language models, caches can quickly consume gigabytes of space.

LLM Cache Management

```
# List large model files (>1GB)
find ~/.cache/huggingface -type f -size +1G | xargs du -h | sort -hr

# Remove specific large models (examples)
rm -rf ~/.cache/huggingface/hub/models--facebook--opt-350m
rm -rf ~/.cache/huggingface/hub/models--mistralai--Mistral-7B-v0.1

# Keep track of which models you've deleted
echo "Deleted: facebook/opt-350m on $(date)" >> ~/model_cleanup_log.txt
```

## 7.4 Creating a Cleanup Script

For repeatable cleanup operations, create a maintenance script.

ML Environment Cleanup Script

```bash
#!/bin/bash
# ML Environment Cleanup Script

echo "=== Starting ML Environment Cleanup ==="
echo "Date: $(date)"

# Check initial disk usage
echo "Initial disk usage:"
df -h | grep /home

# Clear pip cache
echo "Clearing pip cache..."
pip cache purge -y

# Clear conda packages
echo "Clearing conda package cache..."
conda clean -a -y

# Remove specific large models (customize as needed)
echo "Removing unused model caches..."
rm -rf ~/.cache/huggingface/hub/models--facebook--opt-350m
rm -rf ~/.cache/huggingface/hub/models--google--gemma-2b

# Remove temporary files
echo "Cleaning temporary files..."
find /tmp -user $(whoami) -type f -mtime +7 -delete

# Check final disk usage
echo "Final disk usage:"
df -h | grep /home

echo "=== Cleanup Complete ==="
```

# 8 Persistent Training Sessions

## 8.1 Using Screen for Long-Running Processes

Managing long-running training processes with Screen utility.

Screen Session Management

```bash
# Create a new screen session
screen -S session_name

# Start training in the screen session
python training_script.py --config config.yaml

# Detach from screen (without stopping it)
# Press Ctrl+A, then D

# List running screen sessions
screen -ls

# Reconnect to an existing screen session
screen -r session_name

# Kill a screen session
screen -X -S session_name quit
```

## 8.2 Alternative: Using Nohup

Alternative approach using nohup for persistent processes.

Nohup Command for Persistence

```
# Run process that continues after logout
nohup python training_script.py > training.log &

# Check running jobs
jobs

# Bring job to foreground
fg %job_number

# Check process ID
echo $!

# Monitor log file
tail -f training.log
```

# 9 Version Control and Collaboration

## 9.1 Git Operations

Essential Git commands for version control.

Git Workflow Commands

```
# Clone repository
git clone https://github.com/username/repository.git

# Clone to specific directory
git clone https://github.com/username/repository.git custom-name

# Create and switch to new branch
git checkout -b new-branch-name

# Add changes
git add .

# Commit changes
git commit -m "Descriptive commit message"

# Push to remote
git push origin branch-name

# Pull latest changes
git pull origin branch-name

# Check status
git status

# View commit history
git log --oneline
```

## 9.2 Environment Management

Managing Conda environments for reproducible research.

Conda Environment Commands

```
# Initialize Conda (if needed)
source /opt/conda/etc/profile.d/conda.sh

# List available environments
conda info --envs

# Create new environment
conda create -n env_name python=3.10

# Activate environment
conda activate env_name

# Deactivate environment
conda deactivate

# Install packages
conda install package_name
pip install package_name

# Install from requirements file
pip install -r requirements.txt

# Export environment
conda env export > environment.yml
```

# 10 Federated Learning Setup

## 10.1 Federated Training Commands

Commands for setting up and running federated learning systems.

Federated Learning Setup

```
# Start federated server
python federated_training.py -c='configs/cervical.yaml'

# With Flower framework - Server
python federated_training_flower.py --mode server --wandb-key API_KEY

# Running with screen for persistence
screen -S federated_training
python federated_training.py -c='configs/cervical.yaml'
# Detach with Ctrl+A+D
```

# 11 Data Configuration

## 11.1 Dataset Configuration

Setting up dataset configurations for machine learning models.

Dataset Configuration Example

```
{
    "name": "[diabetes]",
    "task_type": "[binclass]", # binclass or regression
    "header": "infer",
    "column_names": null,
    "num_col_idx": [1, 5, 6, 7],  # numerical columns
    "cat_col_idx": [0, 2, 3, 4],  # categorical columns
    "target_col_idx": [8], # target columns (for MLE)
    "file_type": "csv",
    "data_path": "data/diabetes/diabetes.csv"
    "test_path": null,
}
```

# 12 Model Training and Evaluation

## 12.1 Training Commands for Different Models

Commands for training various types of generative models.

Training Pipeline Commands

```
# Tab-DDPM baseline
python main_train.py --dataname diabetes --method tabddpm --mode train

# TabSyn VAE training
python main.py --dataname diabetes --method vae --mode train

# TabSyn diffusion model training
python main.py --dataname diabetes --method tabsyn --mode train

# Complete Tab-DDPM pipeline
python scripts/pipeline.py --config exp/diabetes/ddpm_cb_best/config.toml --train
    --sample --eval
```

# 13 Authentication Tokens

## 13.1 API and Access Tokens

Keeping track of authentication tokens for various services.

Authentication Token Management

```
# GitHub Personal Access Tokens
# Use environment variables to store tokens securely
export GITHUB_TOKEN=your_token_here

# Similarly for other API keys
export WANDB_API_KEY=your_key_here

# Using tokens in curl requests
curl -O -J -L -H "Authorization: token $API_TOKEN" https://api.example.com/
    resource
```

# 14 Appendix: Quick Reference

### 14.1 Common Commands Cheatsheet

| Task | Command |
|---|---|
| SSH Connection | `ssh username@server` |
| File Transfer (Local → Remote) | `scp file.csv username@server:/path/` |
| File Transfer (Remote → Local) | `scp username@server:/path/file.csv ./` |
| GPU Monitoring | `watch -n 1 nvidia-smi` |
| Start Screen Session | `screen -S session_name` |
| Reconnect to Screen | `screen -r session_name` |
| Detach from Screen | `Ctrl+A, D` |
| Clear CUDA Cache | `torch.cuda.empty_cache()` |
| Create Persistent Environment | `conda create -p /path/to/env python=3.10` |
| Activate Persistent Environment | `conda activate /path/to/env` |
| Export Command History | `history > history.txt` |
| Check Disk Usage | `df -h` |
| Find Large Files | `find /path -type f -size +100M \| sort -hr` |
| Clear Pip Cache | `pip cache purge` |
| Remove HF Model Cache | `rm -rf ~/.cache/huggingface/hub/models--model-name` |

## 15 Troubleshooting Common Issues

### 15.1 Out of Disk Space

When encountering "No space left on device" errors during package installation:

**Disk Space Troubleshooting**

1. **Diagnose**: Run `df -h` to check overall disk usage
2. **Identify**: Use `du -h --max-depth=1 ~/ | sort -hr` to find large directories
3. **Clean caches**: Run `pip cache purge` and `conda clean -a -y`
4. **Remove models**: Delete unused model caches with `rm -rf ~/.cache/huggingface/hub/models--unused-model`
5. **Use no-cache**: Install with `pip install --no-cache-dir package_name`
6. **Clean temporary files**: `rm -rf /tmp/username_files/`

### 15.2 Environment Persistence Issues

When conda environments disappear after inactivity:

**Environment Persistence Solutions**

1. **Use path-based environments**: `conda create -p /persistent/path/env_name`
2. **Avoid name-based activation**: Use `conda activate /persistent/path/env_name` (not `conda activate env_name`)
3. **Create activation scripts**: Generate helper scripts that activate the environment by path
4. **Export environment specs**: Keep `environment.yml` and `requirements.txt` for quick recreation
5. **Use persistent storage**: Install environments in directories known to persist
6. **Document installation steps**: Keep a record of all installation commands for reproducibility

### 15.3 CUDA and GPU Issues

When encountering problems with GPU access and CUDA:

**GPU Troubleshooting**

1. **Verify GPU access**: Run `nvidia-smi` to confirm GPU visibility
2. **Check CUDA installation**: Run `nvcc --version` and `python -c "import torch; print(torch.cuda.is_available())"`
3. **Clear GPU memory**: Add `torch.cuda.empty_cache()` before operations
4. **Use correct CUDA version**: Match torch installation to system CUDA version
5. **Set environment variables**: Use `CUDA_VISIBLE_DEVICES` to control which GPUs are used
6. **Monitor memory usage**: Use `watch -n 1 nvidia-smi` to track GPU memory

## 16 Conclusion

Efficient machine learning operations require careful management of computational resources, environments, and workflows. This guide has provided comprehensive instructions for command extraction, persistent environment setup, and system maintenance, specifically addressing common challenges in shared computing environments.

Key takeaways:

- Maintain detailed records of commands for reproducibility
- Use path-based Conda environments in shared systems for persistence
- Implement regular maintenance of cached data, especially from large models
- Monitor and manage disk usage proactively
- Develop clear workflows for common operations

By implementing these best practices, researchers can focus more on scientific innovation rather than troubleshooting technical issues. The time saved through proper environment management, systematic command tracking, and proactive system maintenance compounds over the course of a research project, ultimately accelerating the pace of discovery and model development.

Furthermore, these practices enhance collaboration by making research more reproducible and environments more consistent across team members. When environments are properly documented and persistent, onboarding new team members becomes significantly easier, and knowledge transfer is more efficient.

The field of machine learning is rapidly evolving, with models growing larger and computational requirements increasing. The discipline required to maintain organized workflows, persistent environments, and efficient resource usage will become increasingly valuable as these trends continue. Researchers who implement systematic approaches to their computational infrastructure position themselves for sustained productivity in an increasingly complex field.