

Git, GitHub Actions & CI/CD

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Why Automate?

Manual Process:

1. Write code
2. Run tests locally (maybe?)
3. Commit
4. SSH into server
5. `git pull`
6. Restart service

Automated (CI/CD):

1. `git push`
2. **CI:** Tests run, linters check code.
2. **CD:** If tests pass, auto-deploy to server

GitHub Automation Ecosystem

1. **Git:** Version control (branches, merges).
2. **GitHub API:** Programmatic access to repos (Issues, PRs).
3. **GitHub Actions:** Serverless compute to run workflows.
4. **Webhooks:** Event-driven triggers.

Part 1: GitHub API & PyGithub

Automating the "boring" stuff.

Authentication:

- Use **Personal Access Tokens (PATs)** (Fine-grained).
- Store in `.env` (Never commit!).

```
from github import Github
import os

g = Github(os.getenv("GITHUB_TOKEN"))
repo = g.get_repo("nipunbatra/stt-ai-teaching")

# List open issues
for issue in repo.get_issues(state='open'):
    print(issue.title)
```

Use Case: AI Code Reviewer

Idea: When a PR is opened, fetch the diff, send to LLM, post comment.

```
pr = repo.get_pull(123)
files = pr.get_files()

for file in files:
    # Get the changes
    patch = file.patch

    # Analyze with LLM
    review = llm.review_code(patch)

    # Post comment
    pr.create_issue_comment(f"## AI Review for {file.filename}\n\n{review}")
```

Part 2: GitHub Actions (CI)

Workflows: YAML files in `.github/workflows/`.

Structure:

- **Triggers:** `on: push`, `on: pull_request`
- **Jobs:** Parallel tasks (`test`, `lint`, `deploy`)
- **Steps:** Sequential commands within a job.

Example: CI Pipeline for Python

```
name: CI

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: |
          pip install pytest ruff

      - name: Lint with Ruff
        run: ruff check .

      - name: Run Tests
        run: pytest tests/
```

CI/CD for ML

ML is harder than standard software.

- **Data Tests:** Validate schema (Pydantic/Great Expectations).
- **Model Tests:** Check performance threshold.
- **Hardware:** Need GPU runners? (Self-hosted runners).

Matrix Testing:

Test across python versions / OSs.

```
strategy:  
  matrix:  
    python-version: ["3.9", "3.10", "3.11"]  
    os: [ubuntu-latest, windows-latest]
```

Secrets Management

Never hardcode API keys.

1. Add secret in GitHub Repo Settings -> Secrets.
2. Access in Action:

```
env:  
  OPENAI_API_KEY: ${ secrets.OPENAI_API_KEY }
```

Part 3: Testing Strategy

The Testing Pyramid for AI:

1. **Unit Tests (Most):** Test individual functions (`preprocess_text()`).
2. **Integration Tests:** Test model loading + inference flow.
3. **System Tests:** API endpoints (FastAPI `TestClient`).
4. **Data Tests:** Validate input distributions.

Writing Tests with Pytest

```
# test_model.py
from my_model import predict

def test_prediction_shape():
    output = predict([1.0, 2.0])
    assert "class" in output
    assert output["confidence"] >= 0.0

def test_model_invariance():
    # Adding noise shouldn't flip prediction
    out1 = predict(x)
    out2 = predict(x + epsilon)
    assert out1["class"] == out2["class"]
```

Pre-commit Hooks

Run checks *before* you push.

Prevents bad code from even reaching the repo.

```
.pre-commit-config.yaml :
```

```
repos:  
-   repo: https://github.com/psf/black  
    rev: 23.3.0  
    hooks:  
    -   id: black
```

Workflow: `git commit` -> Black runs -> Formatting fixed -> Commit succeeds.

Summary

1. **GitHub API:** Automate repo management and reviews.
2. **GitHub Actions:** The engine for CI/CD.
3. **Testing:** Essential for robust ML systems (Unit, Data, Model tests).
4. **Pre-commit:** The first line of defense.

Lab: You will build a full CI pipeline that runs tests and uses a custom Action to auto-grade/review PRs.

