Pre-commit Code Quality

Module 3 of 7

Presenter: Vương Cường — Research Assistant, BAI Lab

Introduction

What is Pre-commit?

Pre-commit = **Automated code quality enforcement** before commits

Purpose

- **Consistent code formatting** - Eliminate style debates
- **Early bug detection** - Catch issues before they enter codebase
- **Enforced coding standards** - Maintain professional quality
- **Automated linting and type checking** - Reduce manual review overhead

Why Essential for Research?

- **Team consistency** - Everyone follows same coding standards
- **Reduced review time** - Focus on logic, not formatting
- **Professional codebase** - Publication-ready code quality
- **Prevents bad commits** - Fail fast philosophy

Comparison with Manual Code Review

Approach	Consistency	Speed	Coverage	Human Error	Automation
Manual Review	Inconsistent	Slow	Partial	High	None
IDE Formatting	Per-developer	Fast	Limited	Medium	Partial
CI/CD Only	Good	Delayed	Complete	Low	Full
Pre-commit Hooks	**Perfect**	**Instant**	**Complete**	**None**	**Full**

Benefits Over Alternatives

- **Immediate feedback** - Errors caught at commit time, not in CI
- **Offline capability** - Works without internet connection
- **Zero configuration drift** - Consistent across all developers
- **Comprehensive coverage** - Multiple tools in one configuration

Key Advantage: Pre-commit hooks catch issues before they enter the codebase, preventing accumulation of technical debt.

Core .pre-commit-config.yaml

```yaml
# .pre-commit-config.yaml - covers 80% of research needs
repos:
  # Ruff - Fast Python linter and formatter
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.6.9
    hooks:
      - id: ruff                    # Linting (replaces flake8, isort, etc.)
        args: [--fix]               # Auto-fix issues when possible
      - id: ruff-format             # Code formatting (replaces black)

  # MyPy - Static type checking
  - repo: https://github.com/pre-commit/mirrors-mypy
    rev: v1.11.1
    hooks:
      - id: mypy
        args: [--install-types, --non-interactive]
        additional_dependencies: [types-requests, types-PyYAML]

  # Basic hooks for general file hygiene
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.6.0
    hooks:
      - id: trailing-whitespace    # Remove trailing spaces
      - id: end-of-file-fixer     # Ensure files end with newline
      - id: check-yaml            # Validate YAML files
      - id: check-json            # Validate JSON files
```

Installation and Setup

```
# Install pre-commit (using uvx)
uvx pre-commit install

# Run on all files (one-time check)
uvx pre-commit run --all-files

# Update hooks to latest versions
uvx pre-commit autoupdate
```

Lab Demo with base-research-repo

Step 1: Add Pre-commit to Project

```
cd base-research-repo

# Create .pre-commit-config.yaml file
cat > .pre-commit-config.yaml << 'EOF'
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.6.9
    hooks:
      - id: ruff
        args: [--fix]
      - id: ruff-format

  - repo: https://github.com/pre-commit/mirrors-mypy
    rev: v1.11.1
    hooks:
      - id: mypy
        args: [--install-types, --non-interactive]
```

Step 2: Test on Existing Code

```
# Test pre-commit on all files
uvx pre-commit run --all-files

# Expected output:
# ruff..................................................Passed
# ruff-format...........................................Passed
# mypy..................................................Passed

# If issues found, they'll be auto-fixed or reported
```

Step 3: Experience Automatic Quality Control

```
# Create deliberately poorly formatted Python file
cat > src/test_quality.py << 'EOF'
import os,sys
import numpy as np

def bad_function(x,y):
    result=x+y
    print("Result is:",result)
    return result

# Missing type hints, poor formatting, etc.
EOF

# Now try to commit
git add src/test_quality.py
git commit -m "test: add poorly formatted code"

# Pre-commit will run and fix/block the commit
```

Important Considerations

Configuration Best Practices

Tool Selection Strategy

- **Ruff** - Replaces multiple tools (flake8, isort, black, etc.)
- **MyPy** - Essential for type safety in research code
- **Basic hooks** - File hygiene and format validation
- **Avoid tool overlap** - Don't use multiple formatters

Performance Optimization

```yaml
# Speed up pre-commit execution
repos:
  - repo: local
    hooks:
      - id: ruff-local
        name: ruff
        entry: uvx ruff check --fix
        language: system
        types: [python]
        require_serial: false
```

Team Adoption Strategy

Mandatory for All Lab Repositories

- **Consistent quality** across all projects
- **Onboarding automation** - New members get tools automatically
- **Review efficiency** - Focus on logic, not style
- **Publication readiness** - Code meets academic standards

Handling Pre-commit Failures

```
# If pre-commit blocks a commit:
# 1. Review the suggested fixes
# 2. Accept auto-fixes or make manual corrections
# 3. Re-stage files and commit again

git add .                     # Stage auto-fixed files
git commit -m "your message" # Retry commit
```

Common Issues and Solutions

- **Hook version conflicts** - Use `pre-commit autoupdate` regularly
- **Slow execution** - Configure `repo: local` for frequently used tools
- **Type checking errors** - Add missing type stubs with `additional_dependencies`
- **Team resistance** - Emphasize time savings and consistency benefits

Summary: Pre-commit Code Quality

What We Covered

✅ **Pre-commit fundamentals** - Automated quality enforcement
✅ **Advantages over manual review** - Consistency and speed
✅ **Essential configuration** - Ruff, MyPy, and basic hooks
✅ **Lab-specific setup** - Integration with base-research-repo
✅ **Team adoption strategy** - Mandatory quality standards

Key Takeaways

1. **Pre-commit eliminates code quality inconsistency** across team
2. **Immediate feedback** prevents technical debt accumulation
3. **Professional codebase** ready for academic publication
4. **Reduced review overhead** - focus on logic, not formatting
5. **Tool consolidation** - Ruff replaces multiple quality tools

Impact on Research Workflow

- **Faster code reviews** - Automated quality checks completed upfront
- **Consistent style** - All team members produce uniform code
- **Early error detection** - Type errors and bugs caught immediately
- **Professional standards** - Code quality suitable for publication

Next Steps

➡️ **Module 4: Hugging Face Hub** - Dataset and model storage
➡️ Add pre-commit to all your existing projects
➡️ Configure team-wide quality standards