UV Package Manager

Module 2 of 7

Presenter: Vương Cường — Research Assistant, BAI Lab

Introduction

What is UV?

UV = **Ultra-fast Python package manager** written in Rust

Key Features

- **10-100x faster than pip** - Rust-powered performance
- **Unified tool interface (uvx)** - Run tools without installation
- **Better dependency resolution** - Smarter conflict handling
- **Cross-platform compatibility** - Windows, macOS, Linux

Why Essential for Research?

- **Faster environment setup** - Get coding quickly
- **Reliable dependency management** - Avoid "it works on my machine"
- **Tool execution without pollution** - Clean development environment
- **Improved CI/CD performance** - Faster automated builds

Comparison with Traditional Tools

Tool	Installation Speed	Dependency Resolution	Caching	Unified Interface
pip	Slow	Basic	Limited	No
conda	Very Slow	Good	Yes	No
poetry	Medium	Good	Yes	No
pipenv	Slow	Good	Yes	No
UV	**Ultra Fast**	**Excellent**	**Yes**	**Yes**

Performance Benchmarks

- **UV vs pip**: 10-100x faster installation
- **UV vs conda**: 50x faster environment creation

Environment Management (Daily Usage)

```
# 1. Create and activate virtual environment
uv venv                              # Create .venv folder
source .venv/bin/activate            # Activate (macOS/Linux)
# or
.venv\Scripts\activate             # Activate (Windows)

# 2. Install project dependencies
uv pip install -e .                # Install current project in development mode
uv pip install -r requirements.txt  # Install from requirements file

# 3. Add new packages
uv pip install pandas torch transformers
uv pip install "torch>=2.0"        # With version constraints

# 4. Tool execution without installation
uvx ruff check .                     # Run linter without global install
uvx black --check .                  # Check code formatting
uvx mypy src/                        # Type checking
```

Package Management

```
# Freeze dependencies
uv pip freeze > requirements.txt

# Sync environment (install exact versions)
uv pip sync requirements.txt

# Update packages
uv pip install --upgrade package-name
```

Step 1: Replace pip with UV

```
# Traditional slow way (DON'T DO THIS):
# python -m venv .venv && source .venv/bin/activate
# pip install -r requirements.txt  # Takes 2-5 minutes

# UV fast way (DO THIS):
cd base-research-repo
uv venv                              # Creates virtual environment instantly
source .venv/bin/activate            # Activate environment
uv pip install -e .                 # Install dependencies in 10-20 seconds
```

Step 2: Development Tools with uvx

```
# Install and run development tools without polluting environment
uvx pre-commit install              # Setup code quality hooks
uvx ruff check src/                 # Check code style and errors
uvx black src/                      # Format code automatically
uvx mypy src/                       # Type checking

# Run tools on specific files
uvx ruff check src/data/dataset.py
uvx black --check src/models/       # Check if formatting needed
```

Step 3: Adding New Research Dependencies

```
# Add machine learning packages quickly
uv pip install torch torchvision transformers
uv pip install datasets wandb accelerate
uv pip install "numpy>=1.24,<2.0"  # Version constraints

# Update requirements file
uv pip freeze > requirements.txt
```

Migration from pip/conda

Replacement Commands

```
# OLD pip commands              # NEW uv commands
pip install package             → uv pip install package
pip install -r req.txt          → uv pip install -r req.txt
pip freeze                      → uv pip freeze
pip uninstall package           → uv pip uninstall package
```

Environment Setup Best Practices

- **Always use virtual environments** - Avoid global package pollution
- **Use uvx for one-time tools** - Don't install globally unless necessary
- **Pin versions in requirements.txt** - Ensure reproducible environments
- **Regular updates** - Keep UV itself updated for latest features

Performance Optimization

- **Use uv for CI/CD** - Dramatically faster build times
- **Cache strategy** - UV automatically caches downloaded packages
- **Parallel installation** - UV installs multiple packages simultaneously

Common Pitfalls to Avoid

- **Don't mix UV with pip** in the same environment - stick to one tool
- **Check UV version** - Ensure team uses same UV version
- **Environment activation** - Always activate before installing packages
- **Requirements file format** - UV supports standard pip requirements format

What We Covered

✅ **UV fundamentals** - Ultra-fast Python package manager
✅ **Performance advantages** - 10-100x faster than traditional tools
✅ **Essential commands** - Environment and package management
✅ **Lab-specific workflow** - Integration with base-research-repo
✅ **Migration considerations** - Replacing pip/conda workflows

Key Takeaways

1. **UV dramatically reduces setup time** for research environments
2. **uvx eliminates tool installation overhead** - run without installing
3. **Better dependency resolution** prevents common conflicts
4. **Unified interface** simplifies Python development workflow
5. **Drop-in replacement** for existing pip commands

Impact on Research Workflow

- **Faster onboarding** - New team members setup in minutes, not hours
- **Reduced friction** - Less time fighting dependencies, more time researching
- **Consistent environments** - Better reproducibility across team
- **Improved CI/CD** - Faster automated testing and deployment

Next Steps

➡️ **Module 3: Pre-commit Code Quality** - Automated code standards
➡️ Replace pip with UV in your current projects
➡️ Setup uvx for your development tools