pip vs uv: Streamlining Python Workflows

Speed & Efficiency Comparison

Key Differences

Speed

- uv resolves dependencies 10-100x faster
- Example: 22.8s vs 2.1s for common data science stack
- Subsequent runs even faster with uv's caching

Workflow

pip:
Requires multiple tools (venv, pip, pip-tools)

uv:
Unified interface for all Python tasks

Features

- Built-in dependency locking (uv lock)
- Automatic virtualenv management
- Project scaffolding (uv init)

Example: Installing NumPy + Pandas + Matplotlib

```
# With pip (traditional)
$ time pip install numpy pandas matplotlib
→ 22.8 seconds (cold cache)
```

```
# With uv (modern)
$ time uv add numpy pandas matplotlib
→ 2.8 seconds (cold cache)
```

Installation & Setup

pip (Traditional)

```
# Comes with Python
python -m pip --version
```

• Pre-installed with Python
• Familiar, but slower

uv (Modern)

```
# Requires installation
pip install uv
uv --version
```

• One-time setup
• Written in Rust for performance

Naresh Edagotti

Virtual Environments Setup

pip

```
python -m venv .venv
source .venv/bin/activate # Linux/Mac
.\.venv\Scripts\activate # Windows
```

3 commands, 2-5 seconds

uv

```
uv venv # Creates+activates in one step
```

1 command, <1 second

Package Management

Installing Packages with pip

```
pip install requests pandas
```

- Slow dependency resolution
- No built-in locking

Installing Packages with uv

```
uv add requests pandas
```

- Blazing fast resolution
- Smart caching system

From requirements.txt with pip

```
pip install -r requirements.txt
```

- No version locking
- Need pip-tools for pinning

From requirements.txt with uv

```
uv add -r requirements.txt
```

- Built-in version locking
- Optimized installation order

Advanced Features

Dependency Locking

```
uv lock # Creates uv.lock
uv pin # Pins versions in requirements.txt
```

- Exact version pinning
- Repeatable builds
- No separate tool needed

Project Initialization

```
uv init my_project
```

- Creates structured project layout
- Adds proper packaging files
- Sets up testing directory

Naresh Edagotti

Complete Workflow Example

Modern Python Workflow

```
uv venv # Create & activate venv
uv init my_web_app # Initialize project
uv add fastapi uvicorn # Install packages
uv run main.py # Run the app
```

4 commands

Traditional

```
python -m venv venv
source venv/bin/activate
mkdir my_app
cd my_app
pip install fastapi uvicorn
python main.py
```

6 commands

Why Switch to uv?

- Reduce boilerplate steps
- Eliminate manual environment activation
- Speed up dependency resolution
- Simplify project setup and maintenance

```
pip install uv && uv --help
```

Naresh Edagotti

STATFUSION Ai