🔧 Python Perfect Package Downloader v2.0

Automatically detect and solve Python package conflicts using AI-powered analysis with intelligent web search and continuous learning.

🆕 What's New in v2.0

🔄 Intelligent Retry Loop

- Automatically retries failed installations with learned context
- Each attempt builds on previous failures
- Stops intelligently when solution is found or impossible

🌐 Enhanced Web Search Integration

- AI actively searches for current solutions online
- Finds real-time compatibility matrices
- Discovers recent fixes from StackOverflow, GitHub issues, and official docs
- Verifies solutions against community knowledge

🧠 Continuous Learning System

- Tracks all attempted solutions across sessions
- Never repeats failed approaches
- Learns from multiple analysis attempts
- Builds comprehensive failure context

📊 Multi-Layer Analysis

- Basic analysis for quick diagnostics
- Enhanced analysis with environment context
- Web-verified solution recommendations
- Alternative package suggestions

🎯 Purpose

Analyzes pip install commands to identify and **automatically solve**:

- ✅ Version conflicts between packages

- ✅ Missing dependencies
- ✅ Build errors and system requirements
- ✅ Python version compatibility issues
- ✅ Deprecated package warnings
- ✅ Transitive dependency conflicts

🚀 Quick Start

Basic Usage

First, navigate to the project directory:

```bash
cd path/to/your/project
```

Then run:

```bash
python runthis.py --new-env
```

When prompted, enter package names separated by spaces:

```
📦 What do you want to install? tensorflow==2.15.0 numpy==1.26.0 pandas
```

Advanced Usage

With separate virtual environment:

```bash
cd path/to/your/project
python runthis.py --new-env
```

Custom retry attempts:

```bash
```

```
cd path/to/your/project
python runthis.py --max-retries 5
```

Auto-fix mode (skips confirmations):

```bash
cd path/to/your/project
python runthis.py --auto-fix
```

Combined example:

```bash
cd path/to/your/project
python runthis.py --new-env --max-retries 5 --auto-fix
```

🔬 How It Works

Phase 1: Analysis Loop (runthis.py)

```
1. Execute pip install commands
2. Capture output and errors
3. Analyze with AI (basic + enhanced)
4. If conflicts found → Generate solutions
5. If solved → Success! ✅
6. If failed → Loop with learned context
7. Trigger web search on repeated failures
8. Max retries reached → Pass to solution.py
```

Phase 2: Resolution Loop (solution.py)

```
1. Load all analysis attempts
2. Try automated solutions sequentially
3. If solution fails → AI web search consultation
4. Get web-verified alternative solutions
5. Try new solutions with verification
6. Continue until solved or max iterations
7. Save comprehensive execution log
```

🧪 Test Cases

Easy Conflicts

```bash
# TensorFlow 2.10 requires numpy<1.24
tensorflow==2.10.0 numpy==1.24.0
```

Medium Conflicts

```bash
# Scikit-learn 1.3 needs numpy>=1.23
scikit-learn==1.3.0 scipy==1.11.0 numpy==1.20.0
```

Hard Conflicts

```bash
# Airflow has strict Flask/Werkzeug requirements
apache-airflow==2.5.0 flask==3.0.0 werkzeug==3.0.0
```

Transitive Dependency Hell

```bash
# Multiple ML frameworks with overlapping requirements
tensorflow==2.12.0 torch==2.0.0 jax==0.4.10
```

Python Version Conflicts

```bash
# typing-extensions 3.x won't work on Python 3.11+
django==4.2 python-dateutil==2.8.2 typing-extensions==3.10.0
```

Build Tool Issues

```bash
```

```
# Requires system libraries (OpenSSL, libcurl)
cryptography==41.0.0 pycurl==7.45.2
```

📁 Output Files

Analysis Files (per attempt)

- `exectest1_attempt_N.json` - Basic analysis for attempt N
- `exectest2_attempt_N.json` - Enhanced analysis for attempt N
- `exec_attempt_N.json` - Conversation log (basic)
- `diagnose_attempt_N.json` - Conversation log (enhanced)

Resolution Files

- `resolution_log.json` - Complete execution log with all commands
- `web_search_consultation_N.json` - AI web search findings

History Files

- `llmhist.json` - Complete interaction history across sessions
- `exec.json` - Basic analysis conversations
- `diagnose.json` - Enhanced diagnostic conversations

🔍 Key Features Explained

1. Intelligent Retry with Context

```python
# Each retry includes:
- Previous failure information
- Commands that were already tried
- Error outputs and exit codes
- Web search results from previous attempts

# AI learns what NOT to try again
```

2. Mandatory Web Search on Failures

```python
# After attempt 1 fails, AI MUST:
- Search exact error messages on Google
- Find version compatibility matrices
- Check official documentation
- Look for recent GitHub issues
- Search StackOverflow solutions from 2024-2025
```

3. Solution Verification

```python
# Each solution includes:
- Verification command (e.g., pip check)
- Expected outcome description
- Fallback plan if it fails
- Confidence level (high/medium/low)
- Source reference (URL or documentation)
```

4. Alternative Package Suggestions

```python
# If original packages are problematic:
{
  "original": "problematic-package",
  "alternative": "better-alternative",
  "reason": "Known compatibility issues",
  "compatibility": "Works with Python 3.11+"
}
```

5. Continuous Until Solved

```python
```

```
# Loop structure:
while not solved and iteration < 20:
    try_solutions()
    if all_failed:
        ai_web_search()  # Get new solutions
        continue
    if success:
        verify()
        break
```

📊 Resolution Strategy

Tier 1: Initial Solutions (from JSON)

- Quick fixes based on initial analysis
- Standard version compatibility updates
- Common conflict resolution patterns

Tier 2: Web-Verified Solutions (AI Search)

- Current working version combinations
- Recent community-tested fixes
- Official documentation recommendations

Tier 3: Alternative Approaches

- Different package versions
- Alternative packages entirely
- Environment-level solutions (venv creation)

Tier 4: Deep Analysis

- System dependency checks
- Python version compatibility
- Build tool requirements

🎛️ Configuration Options

runthis.py Options

Flag	Default	Description
`--new-env`	False	Create separate virtual environment
`--max-retries`	3	Analysis retry attempts
`--auto-fix`	False	Skip confirmations

solution.py Options

Flag	Default	Description
`--max-ai-iterations`	8	Max AI consultations
`--no-ai`	False	Disable AI (JSON only)
`--auto-apply`	False	Apply without confirmation
`--log`	resolution_log.json	Log file path

🔧 Advanced Usage Examples

Debug Mode (Maximum Retries)

```bash
python runthis.py --max-retries 10 --auto-fix
```

Conservative Mode (Minimal AI)

```bash
python runthis.py --max-retries 1
python solution.py --max-ai-iterations 2
```

Aggressive Resolution

```bash
python runthis.py --max-retries 5 --auto-fix
# solution.py automatically runs with --auto-apply
```

📚 Understanding the Logs

resolution_log.json Structure

```json
{
  "resolution_status": "SOLVED" | "INCOMPLETE",
  "ai_consultations": 3,
  "web_searches": 2,
  "total_commands": 15,
  "successful_commands": 12,
  "solution_history": [...],
  "execution_log": [
    {
      "command": "pip install package==version",
      "success": true,
      "output": "...",
      "timestamp": "2025-10-05 14:30:00"
    }
  ]
}
```

web_search_consultation_N.json

```json
{
  "web_searches_performed": [
    {
      "query": "tensorflow numpy compatibility",
      "findings": "TensorFlow 2.15 requires numpy<1.27",
      "source_urls": ["stackoverflow.com/..."],
      "relevance": "high"
    }
  ],
  "recommended_solutions": [...]
}
```

🚨 Troubleshooting

"Max retries reached without success"

Solution: Check the analysis files to see what was tried. Often means:

- System dependencies missing (install with apt/brew)
- Python version incompatibility
- Package combinations fundamentally incompatible

"AI consultation unavailable"

Solution: Check that LLM.py is properly configured with API keys.

"All solutions failed"

Solution: Review `resolution_log.json` for patterns. Consider:

1. Creating a fresh virtual environment
2. Updating pip itself: `pip install --upgrade pip`
3. Installing system dependencies
4. Using alternative packages

💡 Pro Tips

1. **Start with flexible versions:**

```bash
# Instead of: tensorflow==2.15.0
# Try: tensorflow>=2.14,<2.16
```

2. **Let the system learn:**
   - Don't interrupt mid-analysis
   - Increase `--max-retries` for complex conflicts
   - Review web search findings for insights

3. **Use virtual environments:**

```bash
python runthis.py --new-env
# Isolates conflicts from system packages
```

4. **Check logs for patterns:**

```bash
# Look for repeated failures
cat resolution_log.json | grep "command"
```

5. **Manual intervention hints:**
   - If AI suggests alternative packages, try them manually
   - Check official docs for breaking changes
   - Consider upgrading Python itself

🤝 What Makes This System Smart

Traditional pip install:

```
pip install package1 package2
❌ Error: conflict detected
→ Manual research required
→ Trial and error
→ Hours of debugging
```

This system:

```
python runthis.py
📊 Analyzes conflicts automatically
🌐 Searches for current solutions
🔄 Tries verified fixes
✅ Resolves automatically
→ Minutes, not hours
```

📈 Success Metrics

The system considers resolution successful when:

- ✅ All packages installed without errors
- ✅ `pip check` passes (no broken dependencies)
- ✅ Verification imports work
- ✅ No conflicts in dependency tree

🎓 Learning Resources

The AI learns from:

- **Official Documentation**: numpy.org, tensorflow.org, etc.
- **Community**: StackOverflow, Reddit, GitHub Issues
- **Package Indices**: PyPI compatibility data
- **Your History**: Previous successes/failures in llmhist.json

---

🔮 Future Enhancements

- [ ] Support for conda packages
- [ ] Parallel solution testing
- [ ] Machine learning on solution success rates
- [ ] Integration with requirements.txt
- [ ] Docker container generation for complex conflicts
- [ ] Automatic system dependency installation

📝 License & Credits

Built with ❤️ for developers tired of dependency hell.

Core Technologies:

- Python subprocess for command execution
- LLM integration for intelligent analysis
- Web search for real-time solutions
- JSON for structured data exchange

Remember: The system gets smarter with each use. The more conflicts it sees, the better it becomes at solving them! 🚀