BioPipelines User Manual

Getting Started

Setup

Important: Always run pipelines from the BioPipelines root directory.

```
cd /path/to/biopipelines
jupyter notebook my_pipeline.ipynb
```

Basic Pipeline Structure

```
from PipelineScripts.pipeline import Pipeline
from PipelineScripts.ligand_mpnn import LigandMPNN
from PipelineScripts.boltz2 import Boltz2

# 1. Create pipeline
pipeline = Pipeline("PipelineName", "JobName", "Description")

# 2. Configure resources
pipeline.resources(gpu="V100", time="24:00:00", memory="16GB")

# 3. Add tools sequentially
tool1 = pipeline.add(ToolClass(parameters))
tool2 = pipeline.add(ToolClass(input=tool1.output, other_params))

# 4. Execute
pipeline.save()           # Generate scripts
pipeline.slurm(email="")  # Submit to cluster
```

Tool Categories

Structure Generation

Generate protein backbones and scaffolds from templates or de novo.

- **RFdiffusion** - Backbone structure generation
- **RFdiffusionAllAtom** - All-atom generation with ligands

Sequence Generation

Design amino acid sequences for generated or existing structures.

- **LigandMPNN** - Ligand-aware sequence design
- **ProteinMPNN** - General protein sequence design

- **MutationComposer** - Mutation-guided generation
- **Fuse** - Protein-protein fusion design

Folding/Cofolding

Predict structures from sequences, with support for ligands and complexes.

- **AlphaFold** - Structure prediction (ColabFold)
- **Boltz** - Advanced prediction with noncovalent interactions

Analysis of MPNN Output

Analyze and characterize sequence design results.

- **MutationProfiler** - Statistical analysis of sequence variations

Structure Analysis

Analyze predicted structures for binding, stability, and other properties.

- **ResidueAtomDistance** - Distance measurements and contacts

Filtering

Process, filter, and combine results from multiple tools and cycles.

- **Filter** - Expression-based result filtering
- **MergeDatasheets** - Combine analysis results
- **ConcatenateDatasheets** - Merge datasets across cycles

- **RemoveDuplicates** - Sequence deduplication

Visualization

Create visual representations and analysis sessions.

- **PyMOL** - Automated molecular visualization sessions

Miscellaneous

Utility tools for pipeline management and data handling.

- **LoadOutput** - Import results from previous pipelines

Core Concepts

Datasheets

All tools communicate through standardized CSV datasheets: - **Structure tools**: `id, structure_file, [metrics]` - **Sequence tools**: `id, sequence, source_id, [scores]` - **Analysis tools**: `id, source_structure, [measurements]`

Tool Chaining

Connect tools through their `.output` attributes:

```
tool1 = pipeline.add(ToolA(input_params))
tool2 = pipeline.add(ToolB(input=tool1.output))  # Uses tool1's output
```

Resource Management

Configure compute resources per pipeline:

```
pipeline.resources(
    gpu="V100",         # GPU type: T4, V100, A100
    memory="16GB",      # RAM allocation
    time="24:00:00",    # Wall time limit
    nodes=1             # Number of compute nodes
)
```

Common Workflow Patterns

Basic Design Cycle

```python
# Structure → Sequences → Prediction → Analysis
rfd = RFdiffusion(pdb="template.pdb", contigs="A1-100")
lmpnn = LigandMPNN(structures=rfd.output, ligand="ATP")
boltz = Boltz2(proteins=lmpnn.output, ligands="SMILES")
analysis = ResidueAtomDistance(structures=boltz.output, residues="A50", atoms="B101")
```

Iterative Optimization

```python
best_structure = initial_structure
for cycle in range(5):
    sequences = LigandMPNN(structures=best_structure, num_sequences=20)
    structures = Boltz2(proteins=sequences.output)
    analysis = ResidueAtomDistance(structures=structures.output, ...)
    filtered = Filter(data=analysis.output, expression="distance < 5.0")
    best_structure = SelectBest(pool=structures.output, datasheets=filtered.output)
```

Multi-Target Analysis

```python
# Analyze multiple conditions and merge results
apo_structures = Boltz2(proteins=sequences.output)
holo_structures = Boltz2(proteins=sequences.output, ligands="ATP")
apo_analysis = ResidueAtomDistance(structures=apo_structures.output, ...)
holo_analysis = ResidueAtomDistance(structures=holo_structures.output, ...)
combined = MergeDatasheets(
    datasheets=[apo_analysis.output, holo_analysis.output],
    prefixes=["apo_", "holo_"],
    calculate={"stability_change": "holo_energy - apo_energy"}
)
```

Environment Configuration

Conda Environments

Tools automatically switch between required environments: - `ProteinEnv` - General protein tools - `Boltz2Env` - Boltz prediction - `ligandmpnn_env` - LigandMPNN/ProteinMPNN - `RFDEnv` - RFdiffusion tools

SLURM Integration

BioPipelines generates optimized SLURM submission scripts with: - Automatic resource allocation - Environment activation - Job dependency management - Log file organization

Troubleshooting

Common Issues

- **Path errors**: Ensure running from BioPipelines root directory
- **Environment issues**: Check conda environment availability
- **Resource limits**: Adjust GPU/memory requirements for your cluster
- **Tool failures**: Check individual tool logs in job output directories

Debug Mode

Add debugging to pipelines:

```
pipeline.debug = True  # Enable verbose logging
pipeline.dry_run = True  # Generate scripts without submission
```

Log Locations

- **Pipeline logs**: `job_folder/pipeline.log`
- **Tool logs**: `job_folder/XXX_ToolName/tool.log`
- **SLURM logs**: `job_folder/slurm-JOBID.out`

Examples

Complete pipeline examples are available in ExamplePipelines/:

- **`ligandmpnn_boltz2.py`** - Basic sequence design and folding
- **`ligandmpnn_boltz2_cycle.py`** - Iterative optimization
- **`rfdaa_ligandmpnn_boltz2.py`** - Complete de novo design pipeline
- **`ligandmpnn_composer_cycle.py`** - Mutation-guided iterative design

Advanced Features

Custom Environments

Specify custom conda environments:

```
tool = pipeline.add(ToolClass(env="custom_env", **params))
```

Resource Optimization

Fine-tune resource allocation:

```
tool = pipeline.add(ToolClass(
    resources={"gpu": "A100", "memory": "32GB", "time": "48:00:00"},
    **params
))
```

Checkpoint Management

Load previous results:

```
previous_run = LoadOutput("/path/to/previous/job/ToolOutputs/tool_output.json")
next_tool = pipeline.add(ToolClass(input=previous_run.output))
```