ConfigDiscovery: LLM-Driven HPC Software Configuration Discovery via Globus Compute

Generated with Claude Code

February 2026

Contents

Abstract

We present ConfigDiscovery, a system that leverages large language models (LLMs) to automatically discover, configure, and validate scientific software installations on high-performance computing (HPC) systems. Using Globus Compute for remote execution, ConfigDiscovery has successfully configured 49 software packages across two DOE Leadership Computing Facility systems: 26 packages on Polaris (NVIDIA A100 GPUs) and 23 packages on Aurora (Intel GPUs). We introduce a Skills abstraction layer that allows users to express computational intent without specifying implementation details, with automatic selection of the best available implementation. We demonstrate the system's capabilities through a multi-fidelity molecular simulation pipeline that chains five different codes together, performing quantum chemistry calculations, machine learning potential training, molecular dynamics, and trajectory analysis.

1 Introduction

Setting up scientific software on HPC systems is notoriously difficult. Each system has unique module environments, compilers, MPI implementations, GPU architectures, and file system layouts. Researchers often spend days or weeks debugging installation issues before they can begin their actual scientific work. This challenge is amplified when targeting multiple HPC systems with different architectures.

ConfigDiscovery addresses this challenge by combining:

- **Large Language Models** (Claude) for reasoning about software dependencies, reading documentation, and generating configuration scripts
- **Globus Compute** for secure remote execution on HPC systems without direct SSH access
- **Iterative refinement** where the LLM observes errors and automatically adjusts configurations
- **Skills abstraction** allowing users to express intent rather than implementation details

The system produces validated YAML configuration files that specify everything needed to run a piece of software: environment modules, conda packages, environment variables, and executable Python functions that perform actual computations.

2 System Architecture

2.1 Overview

ConfigDiscovery consists of four main components:

1. **Discovery Engine**: An LLM-powered agent that researches software requirements, probes the HPC environment, and iteratively develops working configurations
2. **Compute Backend**: Globus Compute integration for executing commands and functions on remote HPC systems
3. **Configuration Schema**: A structured YAML format that captures all aspects of a software configuration
4. **Skills Layer**: An abstraction that maps computational intent to concrete implementations

2.2 Configuration Schema

Each discovered configuration is stored as a YAML file with the following structure:

```yaml
name: software_name
hpc_system: polaris  # or aurora
endpoint_id: <globus-compute-endpoint-uuid>

environment:
  modules: [module1, module2]
  conda_env: environment_name
  conda_packages: [pkg1, pkg2]
  pip_packages: [pkg3, pkg4]
  env_vars:
    VAR_NAME: value
  setup_commands:
    - source activate script

installation:
  steps:
    - installation command 1
    - installation command 2
  verification: |
    command to verify installation works

execution:
  function: |
    def run_software(...):
        # Python function that runs the software
        return {"status": "completed", ...}
  function_name: run_software
  resources:
    nodes: 1
    cores_per_node: 32
    memory_gb: 64

discovery_log:
  discovered_date: '2026-02-12'
  attempts: 15
  notes: 'Discovery notes and observations'
```

Listing 1: Configuration schema structure

2.3 Globus Compute Integration

Globus Compute enables secure, authenticated remote execution on HPC systems. The `ComputeClient` class provides methods for:

- `run_command()`: Execute shell commands remotely
- `run_function()`: Execute Python functions remotely
- `probe_environment()`: Gather system information
- `test_config()`: Validate a complete configuration

All communication is authenticated via Globus Auth, eliminating the need for SSH keys or VPN access.

3 Supported HPC Systems

ConfigDiscovery supports two DOE Leadership Computing Facility systems at Argonne National Laboratory:

Table 1: Supported HPC Systems

System	Architecture	Packages	Scheduler	Conda Location
Polaris	NVIDIA A100 GPUs	26	PBS	User miniconda
Aurora	Intel PVC GPUs	23	PBS	/opt/aurora/25.190.0/...

3.1 Aurora-Specific Configuration

Aurora is an Intel GPU-based exascale system with unique requirements:

1. **Conda Environment**: Aurora uses Intel's conda distribution at `/opt/aurora/25.190.0/oneapi/intel-co`
2. **PBS Scheduling**: Requires specific options including `filesystems=home:flare` and `system=sunspot`
3. **Worker Spin-up Time**: PBS jobs can take 5–10 minutes to start, requiring client-side timeouts of 600+ seconds

4 Configured Software

ConfigDiscovery has configured 49 scientific software packages across both systems. Table 2 summarizes the results.

Summary: Polaris has 26 packages (24 fully tested, 2 require manual setup). Aurora has 23 packages (all fully tested).

4.1 Configuration Details

4.1.1 Quantum Chemistry Codes

xtb (GFN-xTB) Semi-empirical quantum chemistry package installed via conda in a dedicated environment. Configured for geometry optimization, single-point energy calculations, and molecular dynamics. The execution function activates the conda environment via subprocess to ensure proper library loading.

PySCF Python-based quantum chemistry installed via pip. Supports Hartree-Fock, DFT, and post-HF methods (CCSD, MP2). Configured with `OMP_NUM_THREADS` for CPU parallelization.

Psi4 Quantum chemistry suite available as a conda package. Configured for energy calculations with various methods and basis sets.

NWChem Computational chemistry package available via system modules on Polaris and conda on Aurora. Configured for DFT calculations on molecular systems.

Table 2: Software configuration results

Software	**Domain**	**Polaris**	**Aurora**
Psi4	Quantum Chemistry	✓	✓
PySCF	Quantum Chemistry	✓	✓
NWChem	Quantum Chemistry	✓	✓
ORCA	Quantum Chemistry	†	–
CP2K	DFT (GPW method)	✓	✓
GPAW	DFT (PAW method)	✓	✓
Quantum ESPRESSO	Plane-wave DFT	✓	✓
Siesta	DFT (numerical orbitals)	✓	✓
Abinit	DFT (plane-wave)	✓	✓
DFTB+	Tight-binding DFT	✓	✓
xtb	Semi-empirical QM	✓	✓
OpenMM	Biomolecular MD	✓	✓
GROMACS	Classical MD	✓	✓
LAMMPS	Classical MD	✓	✓
NAMD	Biomolecular MD	†	–
AmberTools	Biomolecular Tools	✓	✓
ASE	Atomistic Simulations	✓	✓
MDAnalysis	Trajectory Analysis	✓	✓
Phonopy	Phonon Calculations	✓	✓
SchNetPack	ML Potentials	✓	✓
DeePMD-kit	ML Potentials	✓	✓
MACE	ML Potentials	✓	–[a]
RDKit	Cheminformatics	✓	✓
Open Babel	Molecule Conversion	✓	✓
PyMatGen	Materials Analysis	✓	✓
OpenFOAM	CFD Simulations	✓	✓

✓ = Fully tested   † = Requires manual download   – = Not available

[a]MACE failed to install on Aurora due to disk quota limits

4.1.2 Machine Learning for Chemistry

SchNetPack Neural network potentials for atomistic systems. Installed via pip in a dedicated conda environment with PyTorch and PyTorch Lightning. The execution function generates training scripts that are executed with proper environment activation.

DeePMD-kit Deep learning potentials for molecular dynamics. Installed via pip with TensorFlow backend.

MACE Equivariant message-passing neural network potentials. Available on Polaris; installation on Aurora failed due to disk quota limits during pip installation.

4.1.3 Molecular Dynamics Codes

LAMMPS Large-scale molecular dynamics available via system modules. Configured for various simulation types with proper MPI settings.

OpenMM GPU-accelerated biomolecular simulation installed via conda. Includes built-in test systems for validation.

GROMACS High-performance molecular dynamics. Installed via conda on both systems.

NAMD Biomolecular MD requiring manual download due to license restrictions. Available on Polaris only.

4.1.4 Materials Science

Phonopy Phonon calculations for crystalline materials. Installed via pip with all dependencies including spglib for symmetry analysis.

Quantum ESPRESSO Plane-wave DFT code available via system modules. Configured for electronic structure calculations.

GPAW Real-space DFT code installed via pip. Required special environment variable `MPICH_GPU_SUPPORT_ENABL` to avoid MPI/GPU conflicts.

Siesta, Abinit, DFTB+ Additional DFT codes installed via conda, providing a range of basis set options (numerical orbitals, plane-waves, tight-binding).

5 Skills Abstraction Layer

A major feature of ConfigDiscovery is the Skills system, which provides an abstraction over low-level configurations.

5.1 Motivation

Users want to express computational intent ("compute molecular energy") without specifying implementation details (which code, which system, which parameters). Skills bridge this gap by mapping abstract capabilities to concrete implementations.

5.2 Architecture

```
User Request: "compute energy of this molecule"
        |
        v
+-------------------+
|      Skill        |  molecular_energy
|   (abstract)      |
+--------+---------+
         | selects best implementation
         v
+-------------------+
|     Config        |  configs/aurora/pyscf.yaml
|   (concrete)      |
+--------+---------+
         | executes on
         v
+-------------------+
|   HPC System      |  Aurora via Globus Compute
+-------------------+
```

5.3 Available Skills

Table 3 lists the available skills and their implementations.

Table 3: Available skills and implementations

Skill	**Description**	**Implementations**
molecular_energy	Quantum mechanical energy	Psi4, PySCF, NWChem, xtb, GPAW
geometry_optimization	Optimize molecular geometry	xtb, ASE
molecular_dynamics	Classical/QM MD simulations	LAMMPS, GROMACS, OpenMM, xtb
biomolecular_md	Protein/biomolecule MD	OpenMM, GROMACS, NAMD
trajectory_analysis	Analyze MD trajectories	MDAnalysis
train_ml_potential	Train ML potentials	SchNetPack, DeePMD-kit
ml_potential_predict	Inference with ML potentials	SchNetPack, DeePMD-kit
phonon_calculation	Phonon properties	Phonopy
periodic_dft	Periodic DFT calculations	QE, CP2K, GPAW
cfd_simulation	Computational fluid dynamics	OpenFOAM

5.4 Typed Inputs and Outputs

Skills define typed parameters with units and constraints:

```python
inputs={
    "molecule": ParameterSpec(
        type=DataType.XYZ,
        description="Molecular structure in XYZ format",
        required=True
    ),
    "method": ParameterSpec(
        type=DataType.STRING,
        description="Quantum chemistry method",
        default="HF",
        options=["HF", "DFT", "B3LYP", "CCSD", "MP2", "GFN2-xTB"]
    ),
}
outputs={
    "energy": ParameterSpec(
        type=DataType.FLOAT,
        description="Total energy",
        unit="hartree"
    ),
}
```

Listing 2: Skill parameter specification

5.5 Automatic System Selection

The `SkillExecutor` automatically selects the best available implementation:

```python
from configdiscovery.skills import run_skill

# Automatically picks best available implementation
result = run_skill("molecular_energy", molecule=xyz_string, method="HF")

# Or specify a system
result = run_skill("molecular_energy", molecule=xyz_string, system="aurora")
```

Listing 3: Skill execution

6 Multi-Fidelity Molecular Simulation Pipeline

To demonstrate the power of having multiple codes configured and working together, we developed a multi-fidelity molecular simulation pipeline that chains five different computational methods.

6.1 Pipeline Architecture

The pipeline processes a single molecule through increasing levels of computational sophistication:

1. **Conformer Generation (xtb)**: Generate molecular conformations by perturbing an optimized geometry and computing GFN2-xTB energies for each conformer

2. **Accurate Energy (PySCF)**: Compute Hartree-Fock energy on the lowest-energy conformer for comparison with semi-empirical results

3. **ML Potential Training (SchNetPack)**: Train a SchNet neural network on the conformer dataset to learn the potential energy surface

4. **Molecular Dynamics (xtb)**: Run MD simulation on the molecule using GFN2-xTB
5. **Trajectory Analysis**: Compute RMSD, radius of gyration, and atomic fluctuations from the MD trajectory

6.2 Data Flow

Critically, each step uses output from the previous step—this is not a collection of independent demonstrations but a genuine integrated workflow:

- Step 1 produces conformers and energies → passed to Step 3 for training
- Step 1 identifies the lowest-energy conformer → passed to Step 2 for accurate energy
- Step 1's optimized structure → passed to Step 4 as MD starting point
- Step 4's trajectory frames → passed to Step 5 for analysis

6.3 Implementation

The pipeline is implemented as a Python script that orchestrates remote execution via Globus Compute. Each step is defined as a self-contained function that is serialized and executed on the HPC system:

```python
def run_remote_function(func_code, func_name, endpoint_id, **kwargs):
    compute = get_compute_client(endpoint_id)

    def _run_dynamic(code, fname, fn_kwargs):
        namespace = {}
        exec(code, namespace)
        func = namespace[fname]
        return func(**fn_kwargs)

    result = compute.run_function(
        _run_dynamic, func_code, func_name, kwargs, timeout=1800
    )
    return result.return_value
```

Listing 4: Pipeline execution pattern

Scripts are available for both systems:

- `scripts/multi_fidelity_pipeline.py` (Polaris)
- `scripts/multi_fidelity_pipeline_aurora.py` (Aurora)

6.4 Example: Ethanol Molecule

Running the pipeline on ethanol (C_2H_5OH) demonstrates the complete workflow:

```
python scripts/multi_fidelity_pipeline.py \
    --molecule ethanol \
    --n-conformers 30 \
    --n-epochs 10 \
    --md-steps 500 \
    --temperature 300
```

Listing 5: Running the pipeline

6.4.1 Expected Results

Step 1: Conformer Generation

- 30 conformers generated via geometry perturbation
- Energy range: approximately 50–150 kcal/mol spread
- Computation time: ∼30–60 seconds

Step 2: Ab Initio Energy

- HF/6-31g energy: ∼-154.1 Hartree for ethanol
- Comparison with xtb shows method-dependent energy offset

Step 3: ML Training

- SchNet model trained on 30 conformers (24 train, 6 validation)
- Expected RMSE: 5–20 kcal/mol after 10 epochs
- Saved model can be reused for fast energy predictions

Step 4: Molecular Dynamics

- 500 steps at 300 K (0.25 ps simulation time)
- Trajectory captured as XYZ frames
- Samples thermal fluctuations of the molecule

Step 5: Trajectory Analysis

- RMSD from initial structure: tracks structural deviation
- Radius of gyration: molecular compactness
- RMSF per atom: identifies flexible regions

7 Key Technical Challenges and Solutions

7.1 Globus Compute Timeouts

Problem: Initial tests on Aurora failed with "Execution timed out" errors even though the Globus Compute endpoint was running.

Root Cause: Client-side timeouts were set to 60–120 seconds, but PBS worker jobs need 5–10 minutes to spin up when no workers are idle.

Solution: Use 600-second (10 minute) timeouts for all remote executions:

```
result = compute.run_function(func, timeout=600, **kwargs)
```

Lesson: Don't confuse infrastructure delays (PBS queue time) with execution failures. The Globus Compute endpoint can be "running" while PBS workers are still being scheduled.

7.2 Conda Environment Activation

A major challenge was ensuring that software installed in conda environments could be properly accessed by Globus Compute workers. The workers run in their own environment and don't inherit shell configurations.

Solution: Execution functions write Python scripts to disk and execute them via subprocess with explicit conda activation:

```
conda_activate = "source /path/to/conda.sh && conda activate env && "
cmd = f"{conda_activate}python {script_file}"
subprocess.run(cmd, shell=True, ...)
```

7.3 MPI and GPU Conflicts

Several codes (GPAW, GROMACS) experienced issues with MPI libraries attempting to initialize GPU support on CPU-only nodes.

Solution: Set environment variable `MPICH_GPU_SUPPORT_ENABLED=0` before execution.

7.4 Quote Escaping in Remote Execution

Problem: Nested quotes in shell commands caused syntax errors when serialized through Globus Compute.

Solution: Avoid complex quoting by:

- Writing scripts to temporary files
- Using environment variables instead of inline strings
- Structuring tests with separate command components

7.5 API Version Compatibility

Libraries like Phonopy underwent API changes between versions, causing execution functions to fail with `AttributeError`.

Solution: The LLM discovered and adapted to API changes by consulting documentation and testing different method names (e.g., `set_mesh()` → `run_mesh()`).

7.6 Path and Directory Management

Remote execution occurs in arbitrary working directories. Relative paths caused file-not-found errors.

Solution: Use `os.path.abspath()` for all paths and create output directories explicitly with `os.makedirs(..., exist_ok=True)`.

7.7 License-Restricted Software

Problem: Some packages (ORCA, NAMD) cannot be auto-installed due to license restrictions.

Solution:

- Mark these configs with `manual_download: required: true`
- Provide clear installation instructions in config files
- Use warning indicators in documentation

7.8 Disk Quota Limits

Problem: MACE failed to install on Aurora due to disk quota exceeded during pip installation.

Lesson: Large ML packages with many dependencies can exceed user quotas. Monitor disk usage during installation and consider using shared installations for large packages.

8 CLI Commands

ConfigDiscovery provides a comprehensive CLI for managing configurations and skills:

8.1 Configuration Commands

```
# List available configurations
configdiscovery list

# Show configuration details
configdiscovery show configs/polaris/pyscf.yaml

# Test a configuration (verify + run calculation)
configdiscovery test configs/polaris/pyscf.yaml

# Install dependencies on remote system
configdiscovery install configs/polaris/schnetpack.yaml

# Discover new software configuration
configdiscovery discover "software_name" --endpoint <id> --system polaris

# Run with custom parameters
configdiscovery run configs/polaris/pyscf.yaml \
    --param method=CCSD --param basis=cc-pvdz
```

Listing 6: Configuration CLI commands

8.2 Skill Commands

```
# List available skills
configdiscovery skill list

# Show skill details and implementations
configdiscovery skill show molecular_energy

# Run a skill (auto-selects implementation)
configdiscovery skill run molecular_energy \
    --molecule water.xyz --method HF --basis 6-31g

# Run on a specific system
configdiscovery skill run molecular_energy \
    --molecule water.xyz --system aurora

# Search skills by capability
configdiscovery skill search "energy"
```

Listing 7: Skill CLI commands

9 Conclusions

ConfigDiscovery demonstrates that LLMs can effectively automate the challenging task of configuring scientific software on HPC systems. Key achievements include:

- **49 validated configurations** across two HPC systems (26 on Polaris, 23 on Aurora)
- **Multi-architecture support** spanning NVIDIA and Intel GPU systems
- **Skills abstraction layer** enabling intent-based execution with automatic implementation selection
- **Fully automated discovery** requiring no manual SSH access
- **Reproducible configurations** stored as version-controllable YAML files
- **Integrated workflows** demonstrated through the multi-fidelity pipeline

The few packages requiring manual setup (ORCA, NAMD) have external license restrictions—not limitations of the approach itself.

9.1 Future Work

- **Additional Systems**: Extend to Frontier (ORNL), Perlmutter (NERSC)
- **Container Generation**: Auto-generate Singularity/Apptainer containers from configurations
- **Workflow Integration**: Direct integration with Parsl, Prefect, Globus Flows
- **Skill Chaining**: Declarative multi-step workflow definitions
- **Cost Estimation**: Predict compute hours before execution

Acknowledgments

This work used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy Office of Science user facility at Argonne National Laboratory. ConfigDiscovery was developed using Claude, Anthropic's AI assistant.

A Available Molecules in Pipeline

The multi-fidelity pipeline includes the following pre-defined molecules:

Molecule	**Formula**	**Atoms**
Water	H_2O	3
Methane	CH_4	5
Formic acid	HCOOH	5
Ethanol	C_2H_5OH	9

Custom molecules can be added by providing XYZ-format coordinate strings.

B Configuration Counts by Domain

Domain	**Polaris**	**Aurora**
Quantum Chemistry	5	4
DFT Codes	6	6
Molecular Dynamics	4	3
ML Potentials	3	2
Analysis Tools	4	4
Materials Science	2	2
Engineering (CFD)	1	1
Total	**26**	**23**

C Installation Methods

Method	**Package Count**
Conda (conda-forge)	18
Pip	14
System Modules	6
Manual Download	2