GAT: A High-Performance Rust Toolkit for Power System Analysis

Grid Analysis Toolkit Contributors
https://github.com/monistowl/gat

November 2024

Abstract

We present the Grid Analysis Toolkit (GAT), an open-source command-line toolkit for power system modeling, analysis, and optimization implemented in Rust. GAT provides industrial-grade tools for DC and AC power flow, optimal power flow (OPF), N-1 contingency analysis, state estimation, and reliability assessment. The toolkit emphasizes reproducibility through consistent Arrow/Parquet output formats and Unix-style composability. We validate GAT's DC-OPF and SOCP relaxation solvers against the complete PGLib-OPF benchmark suite of 68 test cases ranging from 5 to 30,000 buses. Results demonstrate 96–97% convergence rates with objective gaps of 4–6% versus AC-OPF baselines. DC-OPF achieves average solve times under 1 second, while the SOCP relaxation provides tighter bounds at the cost of longer computation. GAT is available under an open-source license at `https://github.com/monistowl/gat`.

1 Introduction

Power system analysis tools are fundamental to grid planning, operations, and research. Traditional tools often require proprietary licenses, complex installation procedures, or heavyweight runtime environments. The Grid Analysis Toolkit (GAT) addresses these limitations by providing a fast, reproducible, and composable command-line interface for common power system computations.

GAT is implemented in Rust, a systems programming language that combines C-level performance with memory safety guarantees. This choice enables:

- Single-binary deployment without runtime dependencies
- Predictable memory usage without garbage collection pauses
- Safe parallelism for batch computations
- Cross-platform compatibility (Linux, macOS, Windows)

The toolkit follows Unix design principles: each command performs a single function well, outputs are standardized for interoperability, and commands can be composed through pipelines and scripts. All outputs use Apache Arrow/Parquet formats, enabling seamless integration with modern data science tools including Polars, DuckDB, Pandas, and Apache Spark.

2 Related Work

Several open-source power system analysis tools exist. MATPOWER [1] provides MATLAB-based AC/DC power flow and OPF solvers with an extensive library of test cases. PowerModels.jl [2] implements optimization-based power flow in Julia with support for multiple mathematical formulations. pandapower [3] offers Python-based analysis with an emphasis on usability. PyPSA [4] focuses on large-scale energy system modeling.

GAT differentiates itself through its implementation language (Rust), deployment model (single binary), and output format philosophy (Arrow/Parquet). Rather than competing with existing tools, GAT aims to provide a complementary option for users who prioritize speed, reproducibility, and minimal dependencies.

3 Architecture

3.1 Core Components

GAT is organized as a Rust workspace with modular crates:

`gat-core` Grid types, DC/AC solvers, contingency analysis, state estimation

`gat-io` Data format support (MATPOWER, PSS/E, CIM, Arrow, Parquet)

`gat-algo` Advanced algorithms and solver backends

`gat-cli` Command-line interface and dispatcher

`gat-batch` Parallel job orchestration

`gat-scenarios` Scenario definition and materialization

3.2 Solver Backends

GAT supports multiple optimization backends through the `good_lp` abstraction layer:

- **Clarabel** (default): Pure Rust interior-point solver
- **HiGHS**: High-performance dual simplex and interior-point
- **CBC**: COIN-OR branch-and-cut
- **IPOPT**: Interior-point for nonlinear problems

3.3 Data Formats

Input formats supported:

- MATPOWER (`.m`, `.raw`) with native Rust parser
- PSS/E (`.raw`, `.dyr`)
- CIM/XML (Common Information Model)
- Arrow/Parquet for preprocessed grids

All outputs use Apache Parquet with consistent schemas, enabling downstream analysis without format conversion overhead.

4 Command Interface

GAT provides a comprehensive command-line interface organized by domain:

```
# Power flow analysis
gat pf dc grid.arrow --out flows.parquet
gat pf ac grid.arrow --out flows.parquet

# Optimal power flow
gat opf dc grid.arrow --cost costs.csv --out dispatch.parquet
gat opf ac grid.arrow --tol 1e-6 --max-iter 20 --out dispatch.parquet

# Contingency analysis
gat nminus1 dc grid.arrow --out contingencies.parquet

# Batch execution
gat batch opf --manifest scenarios.json --threads 8 --out results/

# Benchmarking
gat benchmark pfdelta --pfdelta-root data/ -o results.csv
gat benchmark pglib --pglib-dir data/ -o results.csv
gat benchmark opfdata --opfdata-dir data/ -o results.csv
```

5 Benchmark Methodology

We validate GAT against three established benchmark datasets that provide reference solutions for power flow and optimal power flow problems.

5.1 PFΔ Dataset

The PFΔ (Power Flow Delta) dataset provides power flow instances with load perturbations and reference bus voltage solutions. Each test case includes:

- Base case network topology (buses, branches, generators, loads)
- Load perturbation scenarios
- Reference voltage magnitudes (V_m) and angles (V_a) from trusted solvers

Validation metrics:

$$\epsilon_{V_m} = \max_i |V_m^{\text{GAT}}{}_i - V_m^{\text{ref}}{}_i| \tag{1}$$

$$\epsilon_{V_a} = \max_i |V_a^{\text{GAT}}{}_i - V_a^{\text{ref}}{}_i| \tag{2}$$

5.2 PGLib-OPF

The Power Grid Library for Optimal Power Flow (PGLib-OPF) [5] provides standardized AC-OPF test cases derived from MATPOWER with:

- Curated network data with realistic parameters
- Baseline objective values from reference solvers
- Test cases ranging from 3 to 30,000+ buses

Validation metric (relative objective gap):

$$\gamma_{\text{obj}} = \frac{|f^{\text{GAT}} - f^{\text{ref}}|}{|f^{\text{ref}}|} \tag{3}$$

5.3 OPFData

OPFData [6] provides over 300,000 solved AC-OPF instances per grid topology with:

- Load perturbations (FullTop configuration)
- Topology perturbations (N-1 line/generator/transformer outages)
- Reference solutions including bus voltages and generator dispatch
- Objective values from PowerModels.jl solvers

The dataset uses a GNN-friendly JSON format with array-based node and edge representations, supporting machine learning research on power systems optimization.

6 Results

6.1 PFΔ Benchmark

Table 1: PFΔ Benchmark Results (IEEE 14-bus)

Case	Contingency	Converged	Solve Time (ms)	Max V_m Error
case14	n (base)	Yes	0.019	0.0
case14	n (base)	Yes	0.005	0.0
case14	n-1	Yes	0.59	0.0
case14	n-1	Yes	0.59	0.0

GAT achieves 100% convergence on all PFΔ test cases with exact agreement to reference solutions (machine precision). Average solve time is under 1 millisecond for the 14-bus test cases.

6.2 PGLib-OPF Benchmark

We evaluated GAT's DC-OPF and SOCP relaxation solvers against the complete PGLib-OPF benchmark suite of 68 test cases ranging from 5 to 30,000 buses. Baseline objective values are from MATPOWER's AC-OPF solver.

Table 2: PGLib-OPF Benchmark Results (Full Suite)

Method	Conv.	Gap (%)	Time (ms)	VM Viol (p.u.)	Flow Viol (MVA)
DC-OPF	65/65	6.16	898	0.06	3946
SOCP	66/66	4.21	39,036	0.45	1941

Key observations:

- **DC-OPF**: Achieves 100% convergence on 65 successfully parsed cases with 6.16% average objective gap vs. AC baseline. DC-OPF is approximately 40× faster than SOCP but ignores reactive power and losses.
- **SOCP**: Achieves 100% convergence on 66 cases with 4.21% average gap. The 0.45 p.u. voltage "violations" represent the SOC relaxation gap (squared voltage variables not exactly equal to voltage squared), not actual constraint violations.

- **Thermal violations**: DC-OPF shows large apparent thermal violations because it computes active power flow P but the limit is on apparent power $S = \sqrt{P^2 + Q^2}$; without reactive power, the comparison is incomplete.
- **Gap comparison**: Our SOCP gap of 4.21% is within the 1–10% range typical for standard SOCP relaxations on PGLib-OPF cases [5].

Table 3: PGLib-OPF Selected Case Results

Case	Buses	DC Gap (%)	SOCP Gap (%)	SOCP Time (ms)
case14_ieee	14	5.81	0.52	15
case118_ieee	118	4.31	1.51	154
case1354_pegase	1354	0.89	4.70	3,609
case2868_rte	2868	3.13	1.35	13,466
case10480_goc	10480	2.95	0.19	115,328

6.3 OPFData Benchmark

Table 4: OPFData Benchmark Results (IEEE 118-bus, 100 samples)

Metric	Value
Samples tested	100
Convergence rate	100%
Mean solve time	0.29 ms
Network size	118 buses, 186 branches, 54 generators
Mean iterations	1

GAT achieves 100% convergence on all OPFData samples with sub-millisecond solve times. The current solver implementation focuses on power flow feasibility; objective value computation (cost minimization) is planned for future releases.

6.4 Performance Summary

Table 5: Aggregate Performance Across Benchmarks

Dataset	Method	Cases	Convergence	Avg. Solve Time
PGLib-OPF	DC-OPF	65	100%	898 ms
PGLib-OPF	SOCP	66	100%	39,036 ms

The SOCP relaxation provides tighter bounds (4.21% vs 6.16% gap) at the cost of longer solve times. For real-time applications requiring sub-second response, DC-OPF remains practical up to approximately 10,000 buses. SOCP is recommended when solution quality is prioritized over speed.

7 Discussion

7.1 Performance Characteristics

GAT demonstrates consistent sub-millisecond solve times for networks up to 118 buses. The Rust implementation provides predictable performance without garbage collection pauses, making it suitable for real-time applications and large-scale batch processing.

Key performance enablers:

- Native MATPOWER `.m` file parser avoiding external dependencies
- Sparse matrix representations using `nalgebra-sparse`
- Parallel benchmark execution via Rayon
- Zero-copy Arrow memory model

7.2 Current Limitations

The current release has several known limitations:

1. **AC-OPF convergence**: The nonlinear AC-OPF solver (IPOPT backend) may require many iterations on challenging cases. Current validation shows convergence on small-/medium cases; large-scale AC-OPF (10,000+ buses) requires further tuning.
2. **Relaxation gaps**: SOCP relaxation gaps vary from near-zero to 10%+ depending on network topology. Tightening techniques (bound strengthening, McCormick envelopes) are planned.
3. **Transformer modeling**: Three-winding transformers and phase shifters have limited support in the current release.

7.3 Reproducibility

All benchmark runs are reproducible through GAT's `run.json` metadata system:

```
# Re-execute a previous benchmark
gat runs resume benchmark_run.json --execute

# Compare results across runs
gat runs diff run1.json run2.json
```

8 Conclusion

GAT provides a fast, reproducible, and composable toolkit for power system analysis. Comprehensive validation against the PGLib-OPF benchmark suite demonstrates:

- **DC-OPF**: 100% convergence on 65 test cases with 6.16% average objective gap and sub-second solve times
- **SOCP relaxation**: 100% convergence on 66 test cases with 4.21% average objective gap, providing tighter bounds consistent with literature expectations (1–10% typical range)
- **Scalability**: Successful convergence on networks from 5 to 10,480 buses

The Rust implementation enables single-binary deployment without complex dependencies, while Arrow/Parquet output formats ensure interoperability with modern data science ecosystems.

Future work includes:

- Nonlinear AC-OPF solver tuning for large-scale convergence
- SOCP bound tightening for reduced relaxation gaps

- GPU-accelerated linear algebra backends for large-scale problems
- Integration with machine learning frameworks for learning-augmented OPF

GAT is available at `https://github.com/monistowl/gat` under an open-source license.

Acknowledgments

We thank the developers of MATPOWER, PowerModels.jl, PGLib-OPF, and OPFData for providing the benchmark datasets and reference implementations that enable rigorous validation of power system analysis tools.

References

[1] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.

[2] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "PowerModels.jl: An open-source framework for exploring power flow formulations," in *2018 Power Systems Computation Conference (PSCC)*, pp. 1–8, IEEE, 2018.

[3] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "pandapower—an open-source Python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, 2018.

[4] T. Brown, J. Hörsch, and D. Schlachtberger, "PyPSA: Python for power system analysis," *Journal of Open Research Software*, vol. 6, no. 1, 2018.

[5] S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris, S. Fliscounakis, S. Greene, R. Huang, et al., "The power grid library for benchmarking AC optimal power flow algorithms," arXiv preprint arXiv:1908.02788, 2019.

[6] L. Piloto, D. Biagioni, P. Graf, J. Karagiannis, and S. Chatterjee, "OPFData: Large-scale datasets for AC optimal power flow with topological perturbations," arXiv preprint arXiv:2406.07234, 2024.