Software paper for submission to the
Journal of Open Research Software

(1) Overview

Title

NØMAD: Lightweight HPC Monitoring and Diagnostics with Machine Learning-Based Failure Prediction

Paper Authors

1. Tonini, João Filipe Riva (corresponding author)

Paper Author Roles and Affiliations

1. Academic Research Computing, University of Richmond, Richmond, VA 23173, USA.
Email: jtonini@richmond.edu
ORCID: 0000-0002-4730-3805

Abstract

NØMAD (NOde Monitoring And Diagnostics) is a lightweight monitoring and predictive analytics tool for computing infrastructure. It collects system metrics (disk, CPU, memory, I/O, GPU) from any Linux system, with optional SLURM [1] integration for job-level analytics. Data is stored in SQLite, and machine learning models predict job failures before they occur. A key innovation is modeling jobs as nodes in a similarity network, where connections represent shared resource usage patterns. Jobs with similar characteristics often share similar outcomes, and the network structure reveals failure-prone regions in the feature space. The tool provides a real-time web dashboard and supports alerts via email, Slack, or webhooks. NØMAD requires no external databases or complex infrastructure, making it suitable for small-to-medium HPC centers and research groups.

Keywords

HPC; high-performance computing; monitoring; machine learning; SLURM; predictive analytics; failure prediction

Introduction

HPC administrators face a persistent challenge: detecting job failures before they impact researchers. Traditional monitoring approaches fall into two categories: enterprise IT solutions and HPC-specific tools, each with distinct trade-offs.

Enterprise monitoring solutions like Prometheus [2], Grafana [3], and Nagios [4] provide mature, feature-rich platforms but require significant infrastructure—database servers, message queues, and dedicated hardware—that may be impractical for small-to-medium HPC centers. These tools target general IT workloads and lack native understanding of HPC concepts such as job schedulers, batch queues, and compute allocations.

HPC-specific tools address this domain gap but introduce their own complexity. TACC Stats [5] provides comprehensive hardware counter data and correlates system metrics with job performance, enabling detailed post-hoc analysis of application behavior. XDMoD [6] offers institutional-scale job accounting and resource utilization reporting with sophisticated visualization capabilities, but its deployment involves multiple database backends and web services. The Lightweight Distributed Metric Service (LDMS) [7] achieves impressive scalability for continuous monitoring on large systems through its lightweight, distributed architecture. These tools excel at retrospective analysis—understanding what happened after a failure—but provide limited support for real-time prediction of failures before they occur.

A complementary approach treats failure prediction as a pattern recognition problem. Jobs that fail often share common characteristics: excessive NFS writes, memory pressure, or inefficient CPU utilization. Machine learning methods have been applied to HPC failure prediction [8], typically using classification models trained on job features. However, these approaches often treat jobs as independent observations, ignoring the structural relationships between jobs with similar resource profiles.

NØMAD addresses these gaps by combining zero-infrastructure deployment with network-based failure prediction. The key insight is that jobs with similar resource usage patterns tend to experience similar outcomes. Rather than treating each job independently, NØMAD models jobs as nodes in a similarity network where edges connect jobs with comparable behavioral fingerprints. This network structure reveals failure-prone regions in the feature space—clusters of jobs that share characteristics associated with poor outcomes—enabling both risk assessment for individual jobs and actionable recommendations for users.

The choice of similarity metric is consequential for network topology. NØMAD uses cosine similarity on Z-score normalized feature vectors, which emphasizes the *shape* of resource usage profiles rather than absolute magnitudes. A job requesting 64GB of memory with 50% utilization has a similar profile to one requesting 8GB with 50% utilization—both represent reasonable memory sizing—even though their absolute consumption differs by an order of magnitude. This approach draws inspiration from network-based methods in biogeography, where similarity metrics identify emergent regions from species distribution patterns [9]. Just as biogeographical networks reveal ecological boundaries from observational data without predefined regions, job similarity networks reveal behavioral boundaries between successful and failing workloads without predefined failure categories. The similarity threshold (default ≥ 0.7) controls network density: higher thresholds create sparser networks with tighter clusters, while lower thresholds reveal broader patterns at the cost of specificity.

NØMAD provides: (1) zero-infrastructure deployment using a single SQLite database with no external services; (2) system-level monitoring that works on any Linux system without requiring HPC software; (3) optional SLURM integration for job-level analytics and predictive alerts; (4) similarity network analysis that learns failure patterns from historical data; and (5) educational analytics that track the development of computational proficiency over time, enabling insights such as "15 of 20 students improved memory efficiency over the semester."

Implementation and Architecture

NØMAD is implemented in Python and follows a modular architecture with four main components (Figure 1):

Collectors gather metrics from standard Linux tools (`iostat`, `vmstat`, `nvidia-smi`, `nfsiostat`)

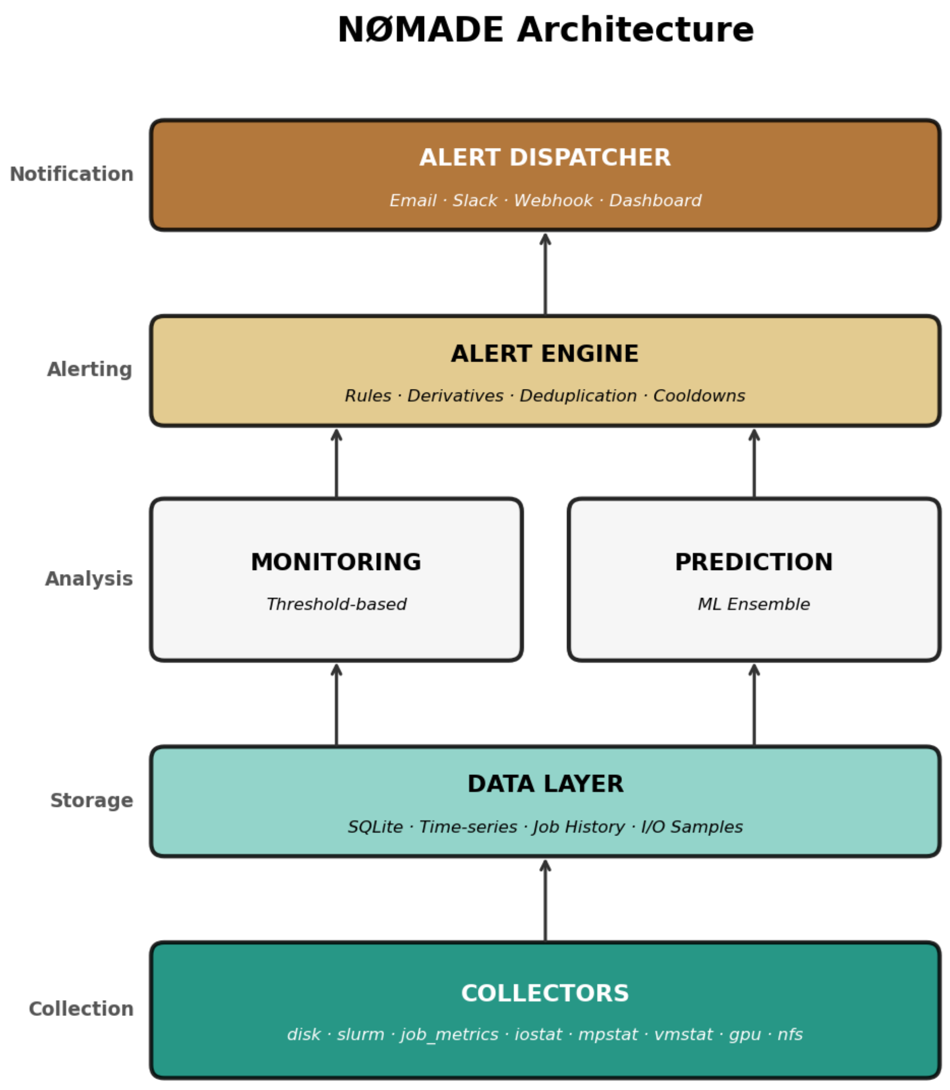

NØMADE Architecture
Notification
ALERT DISPATCHER
Email · Slack · Webhook · Dashboard
Alerting
ALERT ENGINE
Rules · Derivatives · Deduplication · Cooldowns
Analysis
MONITORING
Threshold-based
PREDICTION
ML Ensemble
Storage
DATA LAYER
SQLite · Time-series · Job History · I/O Samples
Collection
COLLECTORS
disk · slurm · job_metrics · iostat · mpstat · vmstat · gpu · nfs

Figure 1: NØMAD system architecture showing the data flow from collectors through the analysis engines to the alert dispatcher and web dashboard. The monitoring engine handles threshold-based alerts while the prediction engine uses ML models for proactive failure detection.

and filesystem utilities. When SLURM is available, additional collectors provide queue state (`squeue`, `sinfo`), job history (`sacct`), and per-job I/O statistics from `/proc/[pid]/io`. Collectors gracefully skip when their requirements are not met (e.g., no GPU monitoring without `nvidia-smi`).

Feature Engineering transforms raw metrics into a feature vector per job. System-level features include I/O wait, memory pressure, swap activity, and device utilization from `iostat`, `mpstat`, and `vmstat`. When SLURM is available, job-specific features (CPU/memory efficiency, NFS write ratios, runtime) extend the vector to 17 dimensions. These features enable similarity-based analysis across jobs using cosine similarity.

ML Prediction uses an ensemble of three models: a Graph Neural Network (GNN) that captures relationships between similar jobs based on cosine similarity of feature vectors; an LSTM that detects temporal patterns and early warning trajectories; and an Autoencoder that identifies anomalous jobs deviating from normal behavior. The ensemble outputs a continuous risk score (0–1) rather than binary classification.

Alert System supports both threshold-based alerts (disk usage, GPU temperature) and predictive alerts using derivative analysis. When the rate of change indicates a threshold will be breached, alerts fire before the actual breach occurs. Notifications route through email, Slack, or webhooks with configurable cooldowns.

Data Readiness Estimator helps administrators determine when sufficient data has been collected for reliable ML predictions. The estimator assesses sample size adequacy (minimum 100 jobs, recommended 500, optimal 1000+), class balance between successful and failed jobs, feature coverage and variance, and data recency. Based on current collection rates, it forecasts time-to-readiness: "At 125 jobs/day, recommended threshold will be reached in approximately 3 days." The command `nomad readiness` produces a detailed report with scores for each dimension and actionable recommendations.

Diagnostics Module provides targeted analysis for common HPC issues. The `nomad diag` commands analyze network performance between nodes (`nomad diag network`), storage health and I/O patterns (`nomad diag storage`), and node-level resource bottlenecks (`nomad diag node`). Each diagnostic produces both summary statistics and detailed drill-down capabilities, helping administrators quickly identify root causes of performance issues.

Infrastructure Monitoring extends beyond compute nodes to cover research workstations and storage systems. The dashboard includes dedicated views for departmental workstations (tracking CPU, memory, disk usage, and logged-in users) and storage servers (monitoring ZFS pool health, NFS client connections, IOPS, throughput, and latency). This holistic view enables administrators to correlate job failures with infrastructure-wide issues such as NFS server congestion or storage capacity constraints.

Web Dashboard

NØMAD provides a real-time web dashboard for monitoring cluster health and job status (Figure 2). The dashboard displays partition-specific metrics including queue depth, node utilization, and resource consumption patterns.

The dashboard includes partition-specific views that allow administrators to monitor different resource types independently (Figure 3). Each partition view displays relevant metrics: CPU utilization and memory pressure for compute partitions, GPU temperature and VRAM usage for GPU partitions, and memory allocation patterns for high-memory partitions.

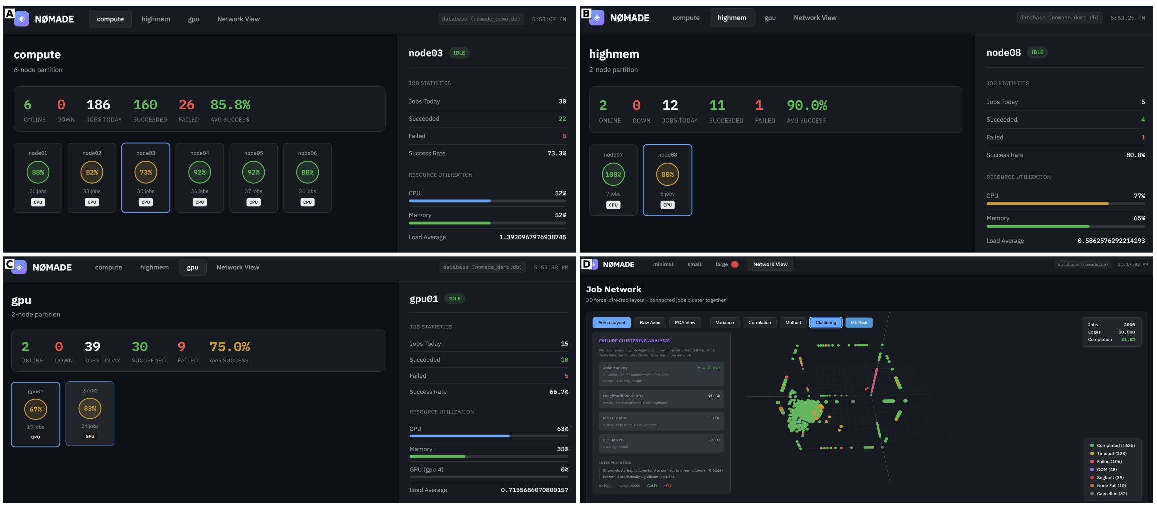


Figure 2: NØMAD web dashboard showing real-time cluster status. Top left displays compute partition metrics, top right shows high-memory partition status, bottom left presents GPU partition health, and bottom right visualizes the job similarity network. The interface provides at-a-glance monitoring for HPC administrators.

Job Similarity Network

A distinguishing feature of NØMAD is the job similarity network visualization (Figure 4). Jobs are represented as nodes in a 3D space defined by their I/O characteristics: NFS write ratio, local write volume, and I/O wait percentage. When full job metrics are available, edges connect jobs with cosine similarity ≥ 0.7 based on their feature vectors.

This visualization reveals how failures cluster in specific regions of the feature space. Jobs in the "danger zone" (high NFS ratio, high I/O wait) show significantly higher failure rates than jobs in the "safe zone" (low NFS ratio, high local writes). Because these patterns emerge from historical data rather than predefined rules, the system can discover site-specific failure modes and provide targeted recommendations to users (e.g., "jobs with your I/O pattern have a 72% failure rate—consider using local scratch instead of NFS").

Quality Control

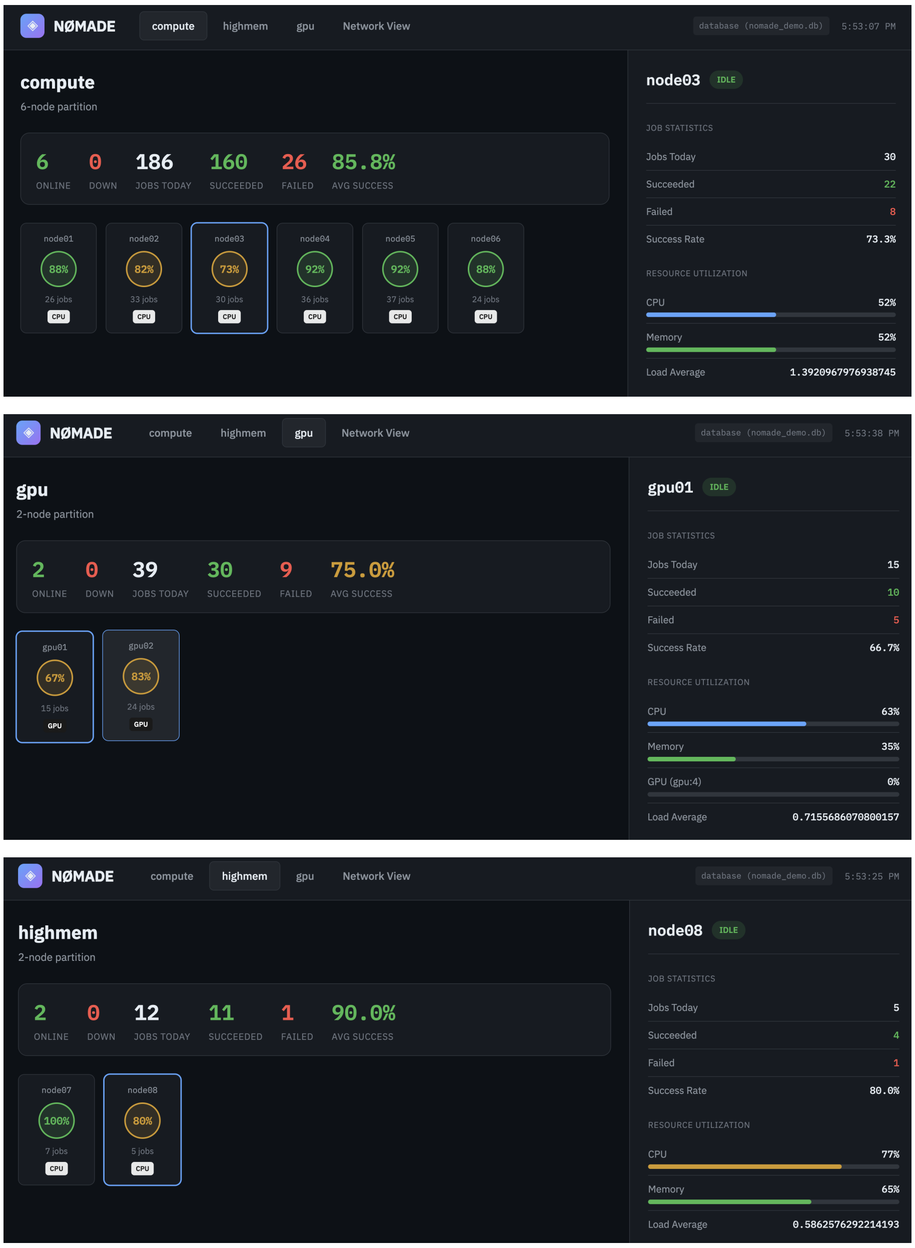
NØMAD includes a comprehensive test suite covering unit tests for collectors, feature engineering, and ML components; integration tests for the full data pipeline; a demo mode (`nomad demo`) that generates synthetic HPC data for testing without requiring a real cluster; and continuous integration via GitHub Actions.

The software has been tested on clusters ranging from 6 to 200 nodes, including both CPU-only and GPU-enabled partitions. The codebase follows PEP 8 style guidelines and includes type hints for improved maintainability.

Educational Analytics

NØMAD includes an educational analytics module (`nomad edu`) designed for classroom instruction, research mentorship, and HPC training programs. The module tracks the development of computational proficiency over time by analyzing per-job behavioral fingerprints across five dimensions: CPU efficiency, memory sizing, time estimation, I/O awareness, and GPU utilization.

Job Explanation (`nomad edu explain`): Translates raw job data into plain-language feedback. For each completed job, the system provides dimension-specific scores (0–100), proficiency levels



compute

6-node partition

6	0	186	160	26	85.8%
ONLINE	DOWN	JOBS TODAY	SUCCEEDED	FAILED	AVG SUCCESS

node01	node02	node03	node04	node05	node06
88%	82%	73%	92%	92%	88%
26 jobs	33 jobs	30 jobs	36 jobs	37 jobs	24 jobs
CPU	CPU	CPU	CPU	CPU	CPU

node03 IDLE

JOB STATISTICS

Jobs Today	30
Succeeded	22
Failed	8
Success Rate	73.3%

RESOURCE UTILIZATION

CPU	52%
Memory	52%
Load Average	1.3920967976938745

gpu

2-node partition

2	0	39	30	9	75.0%
ONLINE	DOWN	JOBS TODAY	SUCCEEDED	FAILED	AVG SUCCESS

gpu01	gpu02
67%	83%
15 jobs	24 jobs
GPU	GPU

gpu01 IDLE

JOB STATISTICS

Jobs Today	15
Succeeded	10
Failed	5
Success Rate	66.7%

RESOURCE UTILIZATION

CPU	63%
Memory	35%
GPU (gpu:4)	0%
Load Average	0.7155686070800157

highmem

2-node partition

2	0	12	11	1	90.0%
ONLINE	DOWN	JOBS TODAY	SUCCEEDED	FAILED	AVG SUCCESS

node07	node08
100%	80%
7 jobs	5 jobs
CPU	CPU

node08 IDLE

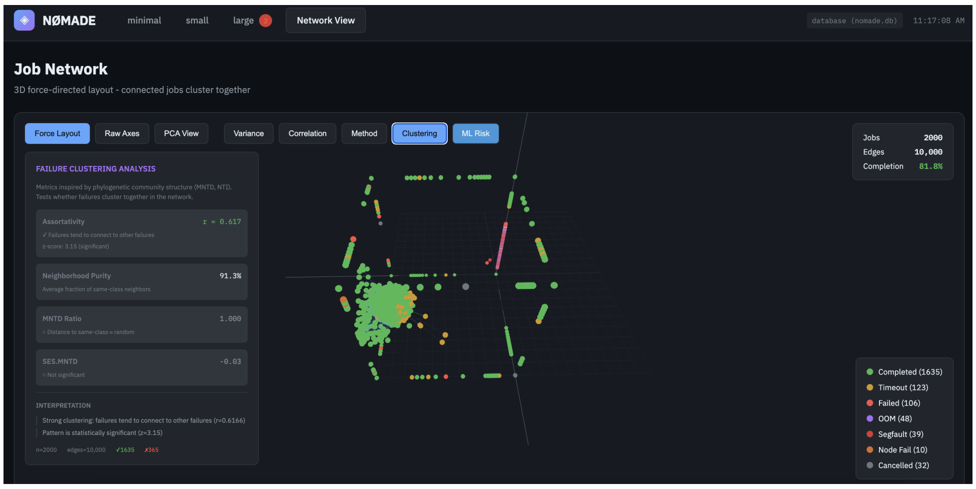
JOB STATISTICS

Jobs Today	5
Succeeded	4
Failed	1
Success Rate	80.0%

RESOURCE UTILIZATION

CPU	77%
Memory	65%
Load Average	0.5862576292214193

Figure 3: Partition-specific dashboard views showing compute (top, 6-node partition with CPU utilization), GPU (middle, 2-node partition with GPU metrics), and high-memory (bottom, 2-node partition optimized for memory-intensive jobs). Each view displays node health rings, job statistics, and resource utilization bars specific to the partition type.



database (nomade.db) 11:17:08 AM

Job Network

3D force-directed layout - connected jobs cluster together

FAILURE CLUSTERING ANALYSIS

Metrics inspired by phylogenetic community structure (MNTD, NTI).
Tests whether failures cluster together in the network.

Assortativity ✓ Failures tend to connect to other failures z-score: 3.15 (significant)	r = 0.617
Neighborhood Purity Average fraction of same-class neighbors	91.3%
MNTD Ratio ○ Distance to same-class ≈ random	1.000
SES.MNTD ○ Not significant	-0.03

INTERPRETATION

Strong clustering: failures tend to connect to other failures (r=0.6166)
Pattern is statistically significant (z=3.15)

n=2000 edges=10,000 ✓1635 ✗365

Jobs	2000
Edges	10,000
Completion	81.8%

● Completed (1635)
● Timeout (123)
● Failed (106)
● OOM (48)
● Segfault (39)
● Node Fail (10)
● Cancelled (32)

Figure 4: Three-dimensional job similarity network visualization. Jobs are positioned by their I/O behavior (NFS writes, local writes, I/O wait) and colored by health score (green = healthy, red = failed). Clustering reveals failure patterns: jobs in the high-NFS, high-I/O-wait region show elevated failure rates compared to jobs using local storage.

(Excellent, Good, Developing, Needs Work), and actionable recommendations. For example, a job with 21% CPU utilization receives the feedback: "Very low CPU utilization— requested 4 cores but used ∼1. Try: `#SBATCH -ntasks=1`."

User Trajectories (`nomad edu trajectory`): Tracks an individual's improvement across their job submissions. The system computes sliding-window averages to identify trends (improving, stable, declining) for each proficiency dimension, enabling mentors to provide targeted guidance.

Group Reports (`nomad edu report`): Aggregates proficiency data across course sections or research groups. Instructors can generate summaries such as "15 of 20 students improved memory efficiency over the semester" or identify that a majority of students struggle with I/O patterns, suggesting curriculum adjustments.

These features address a common challenge in HPC education: students submit jobs that technically complete but exhibit significant resource waste or suboptimal patterns. Traditional approaches require manual inspection of `sacct` output, which is impractical for classes with hundreds of students. NØMAD automates this assessment while providing pedagogically useful feedback that helps students develop genuine computational proficiency rather than simply learning to avoid errors.

Proficiency scores are persisted to the database, enabling longitudinal studies of HPC training effectiveness and research into factors that accelerate skill development.

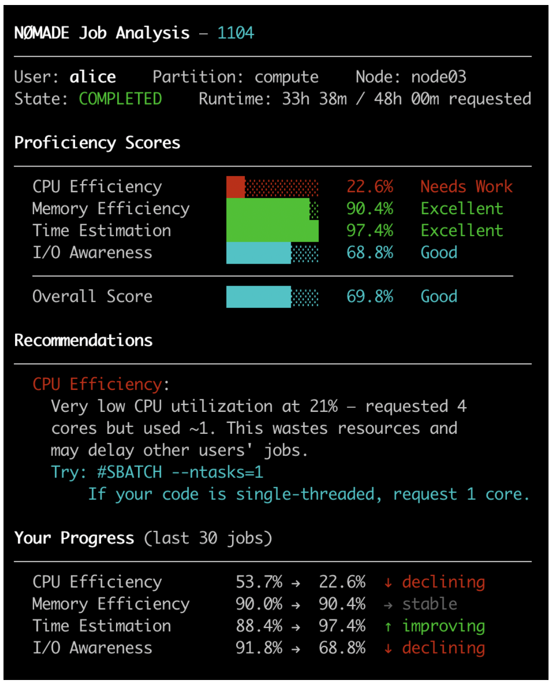
(2) Availability

Operating System

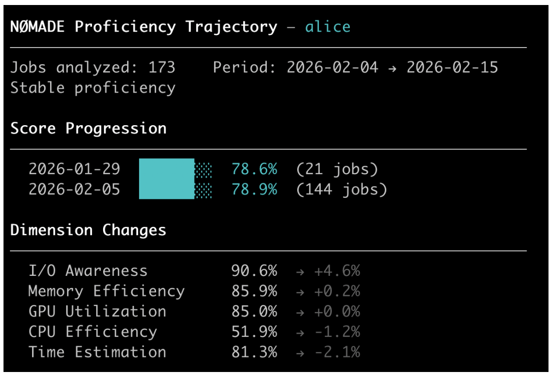
Linux (tested on Ubuntu 22.04, Rocky Linux 9, CentOS 7)

Programming Language

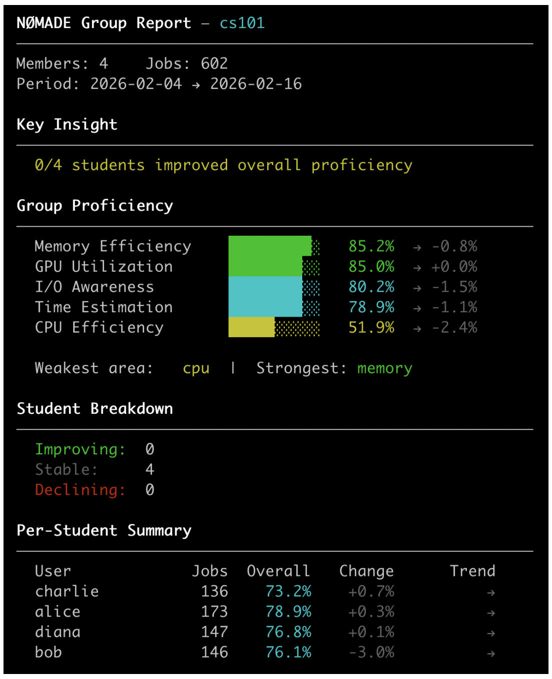
Python 3.9+



```
NØMADE Job Analysis – 1104

User: alice     Partition: compute     Node: node03
State: COMPLETED     Runtime: 33h 38m / 48h 00m requested

Proficiency Scores

  CPU Efficiency                  22.6%   Needs Work
  Memory Efficiency               90.4%   Excellent
  Time Estimation                 97.4%   Excellent
  I/O Awareness                   68.8%   Good

  Overall Score                   69.8%   Good

Recommendations

  CPU Efficiency:
    Very low CPU utilization at 21% – requested 4
    cores but used ~1. This wastes resources and
    may delay other users' jobs.
    Try: #SBATCH --ntasks=1
        If your code is single-threaded, request 1 core.

Your Progress (last 30 jobs)

  CPU Efficiency          53.7% → 22.6%  ↓ declining
  Memory Efficiency       90.0% → 90.4%  → stable
  Time Estimation         88.4% → 97.4%  ↑ improving
  I/O Awareness           91.8% → 68.8%  ↓ declining
```



```
NØMADE Proficiency Trajectory – alice

Jobs analyzed: 173     Period: 2026-02-04 → 2026-02-15
Stable proficiency

Score Progression

  2026-01-29          78.6%  (21 jobs)
  2026-02-05          78.9%  (144 jobs)

Dimension Changes

  I/O Awareness         90.6%  → +4.6%
  Memory Efficiency     85.9%  → +0.2%
  GPU Utilization       85.0%  → +0.0%
  CPU Efficiency        51.9%  → -1.2%
  Time Estimation       81.3%  → -2.1%
```



```
NØMADE Group Report – cs101

Members: 4     Jobs: 602
Period: 2026-02-04 → 2026-02-16

Key Insight

  0/4 students improved overall proficiency

Group Proficiency

  Memory Efficiency              85.2%  → -0.8%
  GPU Utilization                85.0%  → +0.0%
  I/O Awareness                  80.2%  → -1.5%
  Time Estimation                78.9%  → -1.1%
  CPU Efficiency                 51.9%  → -2.4%

  Weakest area:   cpu  |  Strongest: memory

Student Breakdown

  Improving:  0
  Stable:     4
  Declining:  0

Per-Student Summary

  User              Jobs  Overall   Change      Trend
  charlie            136    73.2%    +0.7%          →
  alice              173    78.9%    +0.3%          →
  diana              147    76.8%    +0.1%          →
  bob                146    76.1%    -3.0%          →
```

Figure 5: Educational analytics outputs. **Top left** (`nomad edu explain 1104`): Job explanation with proficiency scores and recommendations. **Top right** (`nomad edu trajectory alice`): User trajectory tracking improvement over 173 jobs. **Bottom** (`nomad edu report cs101`): Group report for a course section with per-student breakdown.

Additional System Requirements

- SLURM workload manager (optional, enables job-level analytics)
- sysstat package (`iostat`, `mpstat`) for system metrics
- Network access for dashboard (default port 5000)
- Minimal disk space for SQLite database ($\sim$1MB per 10,000 samples)

Dependencies

- click $\geq$ 8.0
- toml $\geq$ 0.10
- numpy $\geq$ 1.21
- pandas $\geq$ 1.3
- scipy $\geq$ 1.7
- scikit-learn [10] (optional, for ML features)
- torch (optional, for GNN/LSTM models)
- torch-geometric (optional, for graph neural networks)

List of Contributors

1. Tonini, João Filipe Riva (Lead developer)

Software Location

Archive

Name:	Zenodo
Persistent identifier:	`https://doi.org/10.5281/zenodo.18614517`
Licence:	AGPL-3.0-or-later
Publisher:	João Filipe Riva Tonini
Version:	1.2.2
Date published:	February 2026

Code Repository

Name:	GitHub
Identifier:	`https://github.com/jtonini/nomad-hpc`
Licence:	AGPL-3.0-or-later
Date published:	December 2025

Package Repository

Name:	PyPI
Identifier:	`https://pypi.org/project/nomad-hpc/`
Installation:	`pip install nomad-hpc`

Project Website

URL:	`https://nomad-hpc.com`
Documentation:	`https://jtonini.github.io/nomad-hpc/`

Language

English

(3) Reuse Potential

NØMAD is designed for broad reuse across HPC environments of varying scales. The software can be deployed in several contexts:

Small Research Groups: Groups managing individual workstations or small clusters can use NØMAD's zero-infrastructure design to implement sophisticated monitoring without enterprise tools. The `nomad demo` mode allows evaluation without requiring existing HPC infrastructure.

Medium HPC Centers: University and institutional HPC centers can deploy NØMAD as either a primary monitoring solution or as a complement to existing tools like Grafana or Nagios, adding predictive capabilities to traditional threshold-based alerting.

Research Applications: The similarity network approach provides a framework for studying HPC failure patterns. The network structure, combined with job-level proficiency scores, enables quantitative analysis of resource usage patterns and their relationship to job outcomes.

Community Data Sharing: The `nomad community export` command generates anonymized datasets suitable for cross-institutional research. Export formats include:

- **Job fingerprints**: Feature vectors with hashed user/job identifiers
- **Failure patterns**: Aggregated statistics on failure modes by resource profile
- **Proficiency distributions**: Anonymized skill development trajectories

These exports enable collaborative research on HPC usage patterns without exposing sensitive institutional data. Researchers can study questions such as: Which failure modes are universal vs. site-specific? Do proficiency improvement rates vary by discipline? What resource configurations correlate with job success across different cluster architectures?

The anonymization process removes usernames, job names, and absolute timestamps while preserving the relational structure needed for network-based analysis. Exported data uses relative time offsets and hashed identifiers, making it suitable for publication as supplementary materials or inclusion in shared research datasets.

Educational Use: Beyond the demo mode, NØMAD provides a complete educational analytics pipeline. Instructors can assign Linux groups to course sections (e.g., `cs101`, `bio301`) and use `nomad edu report` to track class-wide proficiency development. Individual students receive automated feedback via `nomad edu explain`, reducing instructor workload while providing immediate, actionable guidance. The module is particularly valuable for:

- **HPC workshops**: Pre/post assessment of participant skills
- **Research onboarding**: Tracking new graduate students' development
- **Classroom instruction**: Automated grading of computational assignments
- **Self-directed learning**: Students can independently review their job history

Extension Points

Developers can extend NØMAD in several ways:

- **Custom Collectors**: Add new metric sources by implementing the collector interface

- **Alert Channels**: Integrate additional notification services beyond email/Slack/webhooks
- **ML Models**: Train site-specific models using the provided feature engineering pipeline
- **Dashboard Widgets**: Extend the web dashboard with custom visualizations

Support

- GitHub Issues: `https://github.com/jtonini/nomad-hpc/issues`
- Email: jtonini@richmond.edu
- Documentation: `https://jtonini.github.io/nomad-hpc/`

Users and developers are welcome to submit bug reports, feature requests, and pull requests via GitHub.

Acknowledgements

The author thanks George Flanagin for advice and inspiration on HPC system administration, and the University of Richmond's Office of the Provost for financial and resource support.

Funding Statement

This work was supported by the University of Richmond.

Competing Interests

The author declares no competing interests.

References

[1] Yoo, A.B., Jette, M.A., Grondona, M. (2003). SLURM: Simple Linux Utility for Resource Management. In: Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol. 2862, Springer, pp. 44–60. DOI: `https://doi.org/10.1007/10968987_3`

[2] Prometheus Authors (2012–present). Prometheus: From metrics to insight. URL: `https://prometheus.io/`

[3] Grafana Labs (2014–present). Grafana: The open observability platform. URL: `https://grafana.com/`

[4] Galstad, E. (1999–present). Nagios: The industry standard in IT infrastructure monitoring. URL: `https://www.nagios.org/`

[5] Evans, R.T., Browne, J.C., Barth, W.L. (2014). Comprehensive resource use monitoring for HPC systems with TACC Stats. In: Proceedings of the First International Workshop on HPC User Support Tools, IEEE, pp. 13–21. DOI: `https://doi.org/10.1109/SC.2014.18`

[6] Palmer, J.T., Gallo, S.M., Furlani, T.R., Jones, M.D., DeLeon, R.L., White, J.P., Simakov, N., Patra, A.K., Sperhac, J., Yearke, T., Rathsam, R., Inber, M., Guillen, O., Cornelius, C.D. (2015). Open XDMoD: A tool for the comprehensive management of high-performance computing resources. Computing in Science & Engineering 17(4):52–62. DOI: `https://doi.org/10.1109/MCSE.2015.32`

[7] Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., Monk, S., Naksinehaboon, N., Ogden, J., Rajan, M., Showerman, M., Stevenson, J., Taerat, N., Tucker, T. (2014). The Lightweight Distributed Metric Service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp. 154–165. DOI: `https://doi.org/10.1145/2616498.2616533`

[8] Tuncer, O., Ates, E., Zhang, Y., Turber, A., Brandt, J., Leung, V.J., Egele, M., Coskun, A.K. (2017). Diagnosing performance variations in HPC applications using machine learning. In: High Performance Computing, Lecture Notes in Computer Science, vol. 10266, Springer, pp. 355–373. DOI: `https://doi.org/10.1007/978-3-319-58667-0_19`

[9] Vilhena, D.A., Antonelli, A. (2015). A network approach for identifying and delimiting biogeographical regions. Nature Communications 6:6848. DOI: `https://doi.org/10.1038/ncomms7848`

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830. URL: `http://jmlr.org/papers/v12/pedregosa11a.html`