

Leveraging AI and Machine Learning for Predicting Job Failures in HPC Systems

Abstract—High-performance computing (HPC) systems are essential for executing complex computations. Simultaneous increase in complexity of HPC systems (e.g., transition to exascale systems with heterogeneous compute devices) and scientific workloads (e.g., complex scientific simulations coupled with artificial intelligence surrogate models) has led to a higher frequency of job failures and, thus, significant performance degradation of HPC systems. This degradation manifests itself at various levels, such as reduced energy efficiency and sustainability, increased power consumption and operational costs, and longer times to run user jobs. We believe that these challenges need to be addressed by deploying intelligent monitoring and alerting systems that can holistically monitor every aspect of operations and report problems early enough so that necessary actions can be taken in a timely fashion to prevent unwanted consequences such as unplanned outages. Previously, we have deployed AIOps [1], a real-time anomaly detection system for telemetry data coming from compute nodes and facility equipment. In this paper, we describe its machine learning-based component that couples with HPC job schedulers and predicts execution status (failure / success) of jobs even before they are scheduled for execution.

Index Terms—Job Failure Prediction, HPC System, Predictive Modeling, Machine Learning, Job Scheduler

I. INTRODUCTION

As high-performance computing (HPC) systems advance toward exascale computing, maintaining their high availability becomes even more challenging due to increasing complexity. Scientific applications have been following similar trend. Their codes become more complex; they need more compute, network bandwidth, and storage to run; they use a diverse set of compute devices such as CPUs and GPUs; they become more heterogeneous internally combining classical HPC primitives (such as MPI-based communication) with newly emerged artificial intelligence (AI) primitives (such as deep learning-based surrogate models that replace simulation loops in classical scientific applications). Increased complexity of HPC systems and HPC and AI applications results in more frequent hardware and application failures leading to increased power consumption, reduced energy efficiency, longer times to run user jobs, and other undesired consequences.

Industry has been addressing some of these challenges by deploying (near) real-time monitoring and anomaly detection systems. In particular, we have implemented a solution [1] within a high-performance cluster management system that identifies anomalies in telemetry data from compute nodes and facility equipment, focusing on the hardware aspect of overall

reliability of HPC systems. Increased complexities in user workloads and underlying hardware platforms has resulted in increased number of job failures due to magnitude of reasons including invalid job configurations, invalid requested hardware resources and bugs in open-source AI software that has become a first-class citizen in HPC environments. We believe that we can address some of these problems even before the job is run.

In this paper, we present our approach to job failure prediction in HPC systems. Concretely:

- We describe machine learning-based approach to predict if a user job will fail or not.
- We describe an end-to-end architecture of the proposed solution that integrates with existing cluster management software and HPC job schedulers.

II. OUR APPROACH

To address the challenges outlined in the previous section, we propose to augment HPC job schedulers with a collection of intelligent services. Figure 1 depicts such a scheduler with 5 services - job failure prediction, cluster health check, anomaly detection for job configuration, predicting execution times and resources (CPU and memory). The job scheduler communicates with these services when users submit their jobs. It first predicts execution time and required resources. It then predicts if there are any anomalies in the job configuration. It then predicts if this job is likely to fail or succeed given its configuration and requested resources. Finally, the job scheduler communicates with the cluster health check to determine which nodes are healthy enough to run the given job. We have already implemented the cluster health check functionality, but have not integrated it yet with job schedulers. Detecting anomalies in configurations and predicting execution time and resources is a future work. In this paper, we focus on the job failure prediction (JFP) service that we have developed, tested, and integrated with the Slurm job scheduler.

In subsequent sections, we describe our machine learning approach and infrastructure to train job failure prediction models (section II-A), Slurm job scheduler components that support JFP functionality (section II-B4), and end-to-end integration of JFP components with the Slurm job scheduler (section II-C).

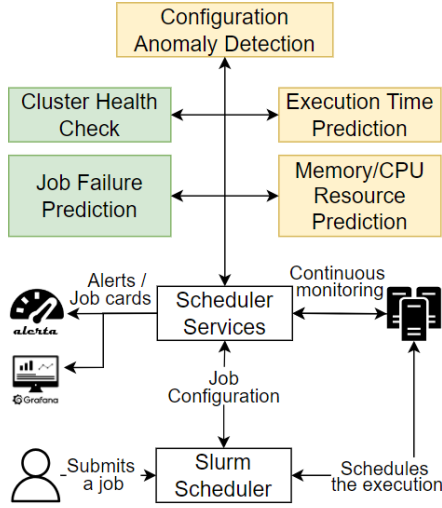


Fig. 1: High-Performance Computing job scheduler integrated with multiple intelligent service to improve operations of data centers.

A. Job Failure Prediction ML Models

1) *Models*: We use gradient-boosted models with tree weak learners implemented in libraries such as XGBoost [2], CatBoost [3] and LightGBM [4]. These models are small in size, easy to train and optimize their hyperparameters, extremely fast at inference phase, naturally suitable for data with missing or unknown feature values (e.g., new users when user ID is an encoded input feature). More importantly, previous research has shown that tree models are state-of-the-art models for tabular data [5] (and job data is tabular data). There are several publications that claim deep learning models (and even deep foundation models) can outperform tree models on certain datasets, but results are questionable and inconclusive: hard or impossible to reproduce, no open-source code available, failed to compare deep learning models with properly tuned tree models, their enormous compute requirement compared to trees.

2) *Datasets*: We have developed several pre-processing and training pipelines that train tree models for different feature sets: (1) generic feature sets available in most HPC systems, (2) restricted feature sets for cases when certain features are missing (such as new user or workload), and (3) standardized feature sets for SWF (standard workload format [6]). Common features include encoded user identifier, submission time, other user details, information about allocated nodes, and other assigned resources. In general, we do not necessarily need special models when input features are not available (since trees can process such inputs by selecting default branches), so this is considered as optional. Some environments use standardized representation of jobs (e.g., SWF), and we support such data sources. However, we advocate against this for multiple reasons. We obtained better results when using features beyond those available in SWF. In addition, it is beneficial to use advanced job and user features available in certain HPC environments. Such types of features are considered private

or sensitive, and are not commonly available in open-source datasets.

3) *Training*: We have adopted the training component of the *REMOVED_FOR_DOUBLE_BLIND_REVIEW* [7] project developed by the authors of this abstract. It targets tree-based models (both boosting and bagging ensembles), uses hyperparameter search at training stages using the Ray Tune [8] and MLflow [9] machine learning platforms.

4) *Workflow*: Figure 2 shows the JFP service running in the presence of job schedulers such as Slurm. In this figure, the components related to Slurm infrastructure are shown in green, and the JFP components are shown in blue. In the following sections, we will provide a detailed description of these components, how they work and interact with each other.

B. Slurm job scheduler

The Slurm [10] cluster resource management solution is quite common in HPC environments. It consists (Fig. 2) of management components ❶ running on the management node, and Slurm daemons (*slurmd*) running on compute nodes ❷. Not all management components are mandatory. In basic installations, only mandatory management daemon *slurmctld* ❸ is present. To enable better job accounting and store job historical data, the database daemon *slurmdbd* ❹ needs to be installed. And to enable third-party application access to Slurm data, the REST daemon *slurmrestd* ❺ needs to be installed. In the following paragraphs we briefly describe these three components, and explain how JFP interacts with the Slurm system using Slurm APIs through the REST Slurm daemon.

1) *Basic Slurm Installation (Scheduler Only)*: This is the minimal working configuration of the Slurm that includes the scheduler *slurmctld* ❸, and compute node daemons *slurmd* ❷. Users submit their jobs, and Slurm runs these jobs on compute nodes in the Slurm cluster. Limited information about submitted and running jobs is available in management daemon through Slurm CLI tools, but no historical information is stored, and job accounting capabilities are limited. Commands like *sacct* and *sreport* will not provide meaningful output due to the absence of the accounting backend (database daemon).

2) *Slurm with Accounting Support (SlurmDBD Enabled)*: This configuration adds the database daemon *slurmdbd* ❹ and a backend Slurm database ❻ (e.g., MariaDB), enabling detailed job accounting. It allows tracking of job history, usage statistics, and enforcement of fair-share or billing policies. This is essential for production clusters where auditability and reporting are required. Tools like *sacct*, *sacctmgr*, and *sreport* become fully functional.

3) *Slurm with REST API and Accounting (Complete Stack)*: This full installation includes management daemon *slurmctld* ❸, Slurm daemons *slurmd* ❷, database daemon *slurmdbd* ❹, and the REST API daemon *slurmrestd* ❺. The presence of API and database daemons enables third-party applications such as JFP service access real-time and historical job data via REST APIs using JSON Web Tokens (JWT) or unix sockets. This configuration

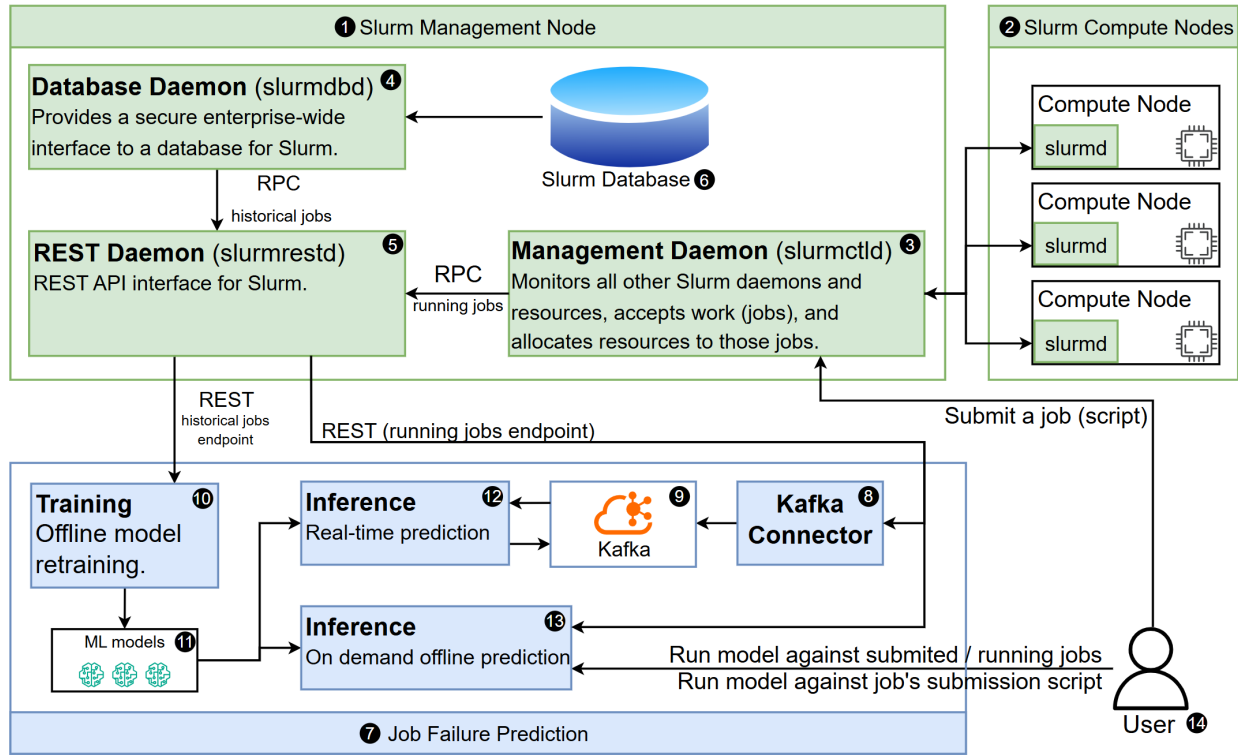


Fig. 2: Job failure prediction (JFP) service workflow.

fully supports automation, monitoring dashboards, and integration with third-party command line tools and services that can interact and manage the Slurm cluster in real time.

By default, Slurm installs only the scheduler *slurmctld* and node daemons *slurmmd*. This configuration supports real-time job submission and execution, but does not support historical job tracking or remote API access. To enable full capabilities, administrators install and configure two additional daemons: database daemon *slurmdbd* for job accounting and REST daemon *slurmrestd* for exposing the REST API interface. Setting up database daemon requires a MariaDB (or MySQL) database to store job history, along with proper database user permissions and a secure configuration file. Each Slurm node, including compute nodes, must have a consistent configuration file pointing to the controller and sharing the same configuration settings such as partitions and authentication method. Authentication across all nodes is handled by *munge* [11], a credential authentication service common in HPC environments, which is installed and started on every machine with an identical *munge* key, typically generated once on the controller and securely copied to all nodes.

For accessing Slurm data through remote REST API, the REST daemon *slurmrestd* 5 is configured to run either on a TCP port or as a Unix socket (e.g., `/run/slurmrestd/slurmrestd.sock`). When JWT-based authentication is used, a Slurm user must be created specifically for the token-based access. This involves running *sacctmgr* to add an account and user entry in the Slurm database, and generating a JWT token for the client using

the JWT key file defined on the controller. A common issue users face is the missing or inconsistent time synchronization between nodes (which breaks *munge*), incorrect file permissions on Slurm and *munge* configuration files, and forgetting to restart or enable Slurm and *munge* daemons. A functional full-stack Slurm installation allows for real-time and historical job control, API-based automation, and detailed usage reporting, but it requires careful orchestration of components across the cluster.

4) *JWT Authentication in Slurm REST daemon*: Slurm REST API *slurmrestd* 5 uses JSON Web Tokens (JWT) to securely authenticate API clients, especially when accessed remotely over HTTP. The tokens are generated from a shared secret key which is usually located in `/var/run/slurm/jwt.key`, and are sent via the `X-SLURM-USER-TOKEN` header. To issue a token, one must first create a Slurm user (e.g., *jwtuser*) and then run: `sudo -u slurm scontrol token username=jwtuser`. This setup ensures controlled, authenticated API usage for automation and dashboards.

C. Job Failure Prediction

The Job Failure Prediction system 7 comprises several components that enable access to Slurm job data (8, 9), train tree-based machine learning models (10, 11), and invoke these models (12, 13) with input job metadata to estimate the likelihood of failure.

1) *Accessing job data*: JFP components access Slurm job data through multiple paths. The training component ⑩ accesses Slurm historical data through the REST daemon ⑤. At inference phase, JFP can function as a real-time ⑫ or on-demand ⑬ failure prediction service. In the former case, JFP requires job data to be available in the messaging systems such as Apache Kafka [12] ⑨. This implies that there must be another service - kafka connector ⑧ - that pulls data from the REST daemon ⑤ and sends it to the Kafka instance. Kafka connector is not part of JFP, and is usually part of a cluster management software. On-demand predictions are made using job data extracted from user job configuration files, or through the REST daemon ⑤.

2) *Training*: Training component ⑩ is responsible for training failure prediction models. As it was mentioned, we use tree-based models, and every training session includes mandatory hyperparameter search. Models of this category are not compute intensive, so model retraining does not require significant compute resources. Training pipelines consist of several components - data retrieval, feature extraction, training with hyperparameter search, and model testing. The data retrieval component communicates with the Slurm REST daemon (5) to retrieve job historical data (through the `/slurmdb/v0.0.42/jobs` API endpoint). That is why JFP requires Slurm to be configured with REST API and database (`slurmdbd`) daemons.

3) *Inferencing*: Inferencing components (⑫ , ⑬) provide real-time on on-demand job failure prediction services. Real-time prediction requires Slurm job data be available in Kafka messaging system ⑨. JFP monitors for new jobs, runs appropriate models ⑪, and sends results back to Kafka in real time so that other components (if they present) can take appropriate actions. JFP can be invoked on demand by users ⑭. In this case, it retrieves information on submitted or running jobs directly through the Slurm REST API. Looking forward, an alternate model may allow users to run JFP locally by extracting job metadata directly from their own `sbatch` job submission scripts before submission, enabling lightweight, decentralized predictions.

III. EXPERIMENTS

The effectiveness of the proposed solution was evaluated using three open-source datasets available from the Hebrew University's parallel workload archive [13].

Netbatch. The workload logs from Intel Netbatch, a grid computing system provided by Intel. The Netbatch grid consists of multiple physical pools, each containing a varying number of nodes, typically numbering in the tens of thousands. **CTC SP2**. The workload logs from the Cornell Theory Center (CTC), a high-performance computing facility at Cornell University in Ithaca, New York, USA. The CTC SP2 system includes 512 nodes, of which 430 are allocated specifically for executing batch jobs. **SDSC Blue**. The workload logs from the SDSC Blue Horizon supercomputer, which is composed of a total of 144 nodes.

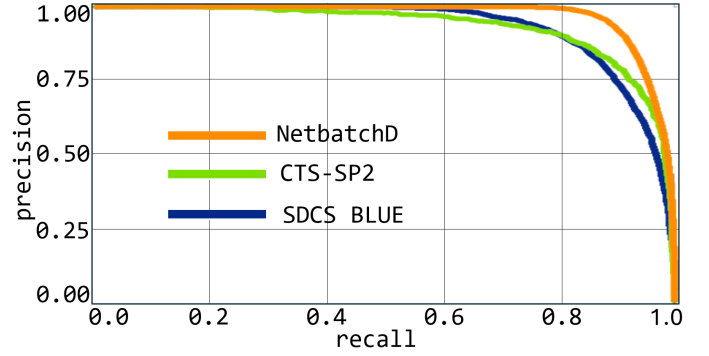


Fig. 3: Precision-Recall (PR) trade-offs for datasets in Table I for various decision thresholds.

In our experiments, we consider each dataset in isolation from the others. Job information is usually considered to be sensitive, what prevents its sharing and building one, source-independent, job failure prediction model. Instead, we build one model for each dataset. We do proper time-aware split of the raw datasets into train and test subsets. Train and test subsets never overlap, and test subsets are always in the "future" compared to the train subsets. This prevents information leakage and enables more robust model evaluation. We used XGBoost library in these experiments, and each training run included hyperparameter search across wide range of parameters using random search algorithm.

These datasets provided a robust foundation for assessing ML model performance across different HPC environments. The results of the JFP experiments conducted on these datasets are presented in Table I. These results show that even with non-sensitive job features available in open-source datasets, accuracy of trained models is better compared to performance of their baseline counterparts (major class classifiers). For the Netbatch dataset, the model accuracy is 97.9% which is a 13.3% improvements compared to the baseline model. The model trained on Cornell Theory Center (CTC_SP2) data demonstrates 93.7% accuracy, a 18.6% improvement compared the baseline model. For the Blue Horizon dataset (SDSC BLUE), the model accuracy is 94.3%, a 19.3% improvement from the baseline model. These performance metrics were computed for decision threshold equal to 0.5.

However, support engineers may have their own preferences regarding false positive and false negative model responses. Figure 3 shows three precision-recall curves for datasets from Table I. This figure shows magnitude of possible model behaviours. Large thresholds result in high precision and low recall, while smaller thresholds result in lower precision and higher recall. For instance, Netbatch model can be configured to have precision close to 1.0 and recall around 0.8 meaning that all positive model responses (e.g., job will fail) are expected to be correct, while 20% of all failed jobs will not be detected. On the other hand, CTS-SP2 and SDSC BLUE models and for the same recall value (0.8) have precision around 0.9.

Dataset	Duration	# Jobs	# Users	Features	Accuracy	Recall	Precision	F1	Baseline Accuracy
NetbatchD	11/12–12/12	261,080	521	13	97.9%	89.7%	97.4%	93.3%	86.4%
CTC-SP2	06/96–06/97	77,222	679	11	93.7%	74.4%	94.2%	83.3%	79.0%
SDSC BLUE	04/00–01/03	32,135	60	12	94.3%	85.5%	89.6%	86.2%	79.0%

TABLE I: Performance of the job failure prediction models on three public datasets.

IV. RELATED WORK

Methods to predict job failures in HPC (and other) systems can be categorized into multiple buckets depending on what life cycle phase the prediction is made (see [14] for introduction to job states) and what features are used. We categorize these approaches into three buckets.

Methods in the first and second bucket detect job failures at submit time (first bucket, right after a user has submitted their job) or run time (second bucket, right before the HPC job scheduler executes the job). Jobs’ features such as user details, job parameters, submission scripts and requested resources are typically available. Works like [14], [15] belong to these categories of methods. From machine learning perspective, researchers have mostly been using tree-based models (decision trees, random forests and gradient-boosted trees). Our approach described in the paper targets job failure prediction at submit and run times.

Methods in the third bucket target job failure prediction at execution time, when an HPC system runs a job. In addition to features available at submit and run times, researchers use features computed based upon system, application and scheduler logs that are usually collected in real time by a cluster management and monitoring software. Papers like [16] address failure prediction at execution phase.

V. CONCLUSIONS

In this paper we have described our approach to predicting job failures in HPC systems. This approach detects possible job failures at submit and run times. It uses gradient-boosted tree models, and integrates with the Slurm HPC job manager. We have demonstrated that the performance of our models exceed performance of baseline models, and support engineers, based on their tolerance of false positive and false negative responses, can fine-tune decision thresholds using receiver operating characteristic (ROC) or precision-recall curves that are automatically created whenever a new version of the model is trained.

We outline the following three key objectives for advancing our system:

- 1) **Predictive Resource Estimation:** Enhance the system’s ability to predict job resource requirements (e.g., memory usage, execution time) to optimize efficiency and reduce energy consumption.
- 2) **Anomalous Job Configuration Detection:** Develop mechanisms to identify unusual or inefficient job configurations to ensure smooth scheduling and stable workflows.
- 3) **AI Agent Copilot Integration:** Integrate with an LLM-based copilot to enable natural language submission and

validation of job scripts. The Job Failure Prediction (JFP) service will be exposed via a REST API, allowing the LLM to query and respond to job status intelligently. This will enable an AI agent chatbot to guide users throughout the job lifecycle via conversational support.

VI. ACKNOWLEDGMENTS

We would like to thank our colleagues from the HPC team for their help and valuable feedback throughout this work. We also appreciate the lab’s team for their contributions and the management team for their continued encouragement and support.

REFERENCES

- [1] “Removed for double-blind review.”
- [2] T. Chen and C. Guestrin, “Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,” p. 785–794, 2016.
- [3] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: unbiased boosting with categorical features,” *Advances in neural information processing systems*, vol. 31, 2018.
- [4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in neural information processing systems*, vol. 30, 2017.
- [5] L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why do tree-based models still outperform deep learning on typical tabular data?” *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.
- [6] “The standard workload format,” <https://www.cs.huji.ac.il/labs/parallel/workload/swf.html>, [Online; accessed 10-August-2025].
- [7] “Removed for double-blind review.”
- [8] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” *arXiv preprint arXiv:1807.05118*, 2018.
- [9] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, “Accelerating the machine learning lifecycle with MLflow,” *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.
- [10] A. B. Yoo, M. A. Jette, and M. Grondona, “Slurm: Simple linux utility for resource management,” in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.
- [11] “MUNGE: an authentication service for creating and validating credentials,” <https://dun.github.io/munge/>, [Online; accessed 10-August-2025].
- [12] N. Garg, *Apache kafka*. Packt Publishing, 2013.
- [13] “Logs of real parallel workloads from production systems,” <https://www.cs.huji.ac.il/labs/parallel/workload/logs.html>.
- [14] A. Banjongkan, W. Pongsena, N. Kerdprasop, and K. Kerdprasop, “A study of job failure prediction at job submit-state and job start-state in high-performance computing system: Using decision tree algorithms [j],” *Journal of Advances in Information Technology*, vol. 12, no. 2, 2021.
- [15] F. Antici, A. Borghesi, and Z. Kiziltan, “Online job failure prediction in an hpc system,” in *European Conference on Parallel Processing*. Springer, 2023, pp. 167–179.
- [16] J.-W. Park, X. Huang, and C.-H. Lee, “Analyzing and predicting job failures from hpc system log,” *Journal of supercomputing*, vol. 80, no. 1, 2024.