femington

Int. J. Intell. Syst. Data Sci. (2026) 1(2):1–10
Article ID: IJISDS-2026-490

From YAML to Queue: DevOps Automation for Reproducible Benchmarks

Jayantjaishwin Shanmugam
Kavitha
jayant.jaishwin4@gmail.com
Independent Researcher

Abstract

This paper reframes scientific benchmarking as "experiments-as-code," presenting a DevOps automation approach that codifies multi-step runs, hyperparameter sweeps, and cross-site portability in declarable runbooks executed against heterogeneous compute tiers. A pair of complementary Python-based orchestration stacks demonstrates how containerized tasks, scheduler adapters, and templated workflows enable continuous, repeatable experiment execution from laptops to leadership-class clusters and public cloud. The approach emphasizes Git-driven configuration, immutable artifacts, and environment capture to strengthen provenance and FAIR-aligned reproducibility, while policy-aware submission integrates with batch systems and secure remote access patterns. Iteration-first specifications (loops, arrays, and conditional stages) simplify large ensembles beyond DAG-only models, and cost-aware planning supports pragmatic cloud bursts without sacrificing portability. The result is a practical blueprint for applying CI/CD-style discipline to versioned configurations, automated provisioning, template reuse, and consistent reporting to scientific benchmarking at scale. By aligning workflow definition, execution, and evidence collection with DevOps practices, the framework reduces operational toil, shortens feedback cycles for model and system tuning, and promotes shareable templates that accelerate onboarding and collaboration across research teams.

Keywords

• Workflow Orchestration • Benchmarking • DevOps • High-performance computing(HPC) • Cloud Computing

1. Introduction

Scientific benchmarking has traditionally been a manual, ad-hoc process, often involving custom scripts, handwritten job submissions, and one-off data collection procedures. This approach is not only time-consuming but also prone to errors and difficult to reproduce. As computational experiments grow in complexity, spanning hyperparameter searches, multi-stage pipelines, and heterogeneous hardware, the need for a more systematic, automated methodology becomes critical. In this paper, we propose a DevOps-inspired approach to benchmarking, which we term "experiments-as-code." This paradigm treats benchmark workflows as version-controlled, declarative specifications that can be automatically executed, monitored, and reproduced across diverse computing environments.

The core idea is to leverage modern software engineering practices such as continuous integration, infrastructure-as-code, and immutable artifacts to manage the entire lifecycle of a scientific experiment. By codifying experiments in structured formats like YAML, researchers can ensure that every run is consistent,

documented, and repeatable. This is particularly important in high-performance computing (HPC) and AI research, where small changes in environment or configuration can lead to significant variations in results. Our approach also aligns with the FAIR principles (Findable, Accessible, Interoperable, Reusable), promoting open science and collaborative research.

We present two independently developed but complementary tools Cloudmesh and SmartSim that embody this experiments-as-code philosophy. Both frameworks provide Python-based APIs and configuration languages for defining and executing complex workflows, but they differ in their target use cases and underlying architectures. Cloudmesh focuses on multi-site, heterogeneous resource integration, while SmartSim specializes in coupled simulation-AI workflows with in-memory data exchange. Despite these differences, both systems share a common goal: to simplify and automate the process of running scientific benchmarks at scale.

This paper makes four contributions:

1. A DevOps-oriented benchmarking abstraction for HPC/AI workflows that treats experiments as versioned, executable code.
2. An iteration-first workflow specification model that directly supports loops, arrays, and conditional stages, going beyond pure DAG-based representations.
3. A portability layer spanning cloud schedulers and HPC workload managers (SLURM, PBS, LSF) with policy-aware submission and secure remote access.
4. Reproducibility mechanisms aligned with the FAIR principles, including environment capture (containers, lockfiles), immutable artifacts, and Git-driven configuration.

The remainder of this paper is organized as follows. Section 2 outlines the key requirements for a DevOps-driven benchmarking framework. Section 3 describes the architecture and implementation of Cloudmesh and SmartSim. Section 4 presents use cases that demonstrate the effectiveness of our approach. Section 5 discusses the implications of our work and compares it with related research. Section 7 concludes the paper and suggests future directions.

2. Requirements for DevOps-Driven Benchmarking

A DevOps-inspired benchmarking framework must address several key requirements to be effective in scientific environments. First and foremost, it must support a wide range of computing resources, from individual workstations to leadership-class supercomputers and public cloud providers. This includes the ability to interface with different workload managers (e.g., SLURM, LSF, PBS) and authentication mechanisms (e.g., SSH, VPN). The framework should also provide a uniform interface for submitting and monitoring jobs, regardless of the underlying infrastructure.

Another critical requirement is the ability to define and execute iterative experiments, such as hyperparameter sweeps and grid searches. Traditional workflow systems often rely on directed acyclic graphs (DAGs), which are well-suited for pipeline-style workflows but less so for iterative or conditional execution. Our approach emphasizes iteration-first specifications, allowing users to easily define loops, arrays, and dynamic parameter spaces. This is especially important for AI and machine learning benchmarks, where hyperparameter tuning is a common and computationally intensive task [1, 2].

Portability and reproducibility are also essential. Benchmark specifications should be independent of the execution environment, allowing them to be run on different systems without modification. This

requires careful management of software dependencies, either through containerization (e.g., Docker, Singularity) or environment modules. Additionally, the framework should capture detailed provenance information, including software versions, hardware configurations, and execution parameters, to ensure that results can be reproduced and verified.

In this context, *immutable artifacts* refer to container images (e.g., Docker, Singularity), environment lockfiles (e.g., Conda `environment.yml`, Pip `requirements.txt`), workflow manifest YAML files, generated datasets, and raw output logs. All such artifacts are versioned and persisted across runs to guarantee provenance.

Cost awareness is another important consideration, particularly when using cloud resources. The framework should provide tools for estimating and monitoring the cost of experiments, helping researchers make informed decisions about resource allocation. This includes support for spot instances, auto-scaling, and other cost-saving mechanisms. Finally, the framework should promote collaboration and reuse through template repositories and version-controlled configurations, enabling researchers to build on each other's work and avoid reinventing the wheel.

3. System Architecture and Implementation

Our experiments-as-code approach is implemented in two complementary systems: Cloudmesh and SmartSim. Both are open-source, Python-based frameworks that provide high-level abstractions for defining and executing benchmark workflows. In this section, we describe the architecture and key features of each system.

3.1 Cloudmesh

Cloudmesh is a modular, plugin-based framework for managing heterogeneous computing resources. Its core components include the Experiment Executor (EE) and the Compute Coordinator (CC). The EE is responsible for generating and executing parameter sweeps, while the CC coordinates tasks across multiple resources. Workflows are defined in YAML, which provides a human-readable and machine-parseable format for specifying experiments.

The EE uses a template-based approach to generate job scripts for different workload managers. Users define a set of parameters and a job template, and the EE automatically generates and submits the corresponding jobs. This makes it easy to run large-scale hyperparameter searches without writing custom scripts. The CC, on the other hand, manages dependencies between tasks and ensures that they are executed in the correct order. It supports both DAGs and dynamic workflows, allowing for complex, conditional execution paths.

Cloudmesh also includes plugins for provisioning cloud resources, managing VPN connections, and monitoring energy consumption. This makes it a comprehensive tool for end-to-end experiment management. The framework is designed to be extensible, with a simple plugin API that allows users to add new functionality as needed.

3.2 SmartSim

SmartSim is designed for coupled simulation-AI workflows, particularly those that require low-latency data exchange between components. It provides an in-memory datastore (Orchestrator) that allows simulations and AI models to share data without writing to disk. This is especially useful for online inference and reinforcement learning, where traditional file-based I/O would be a bottleneck.

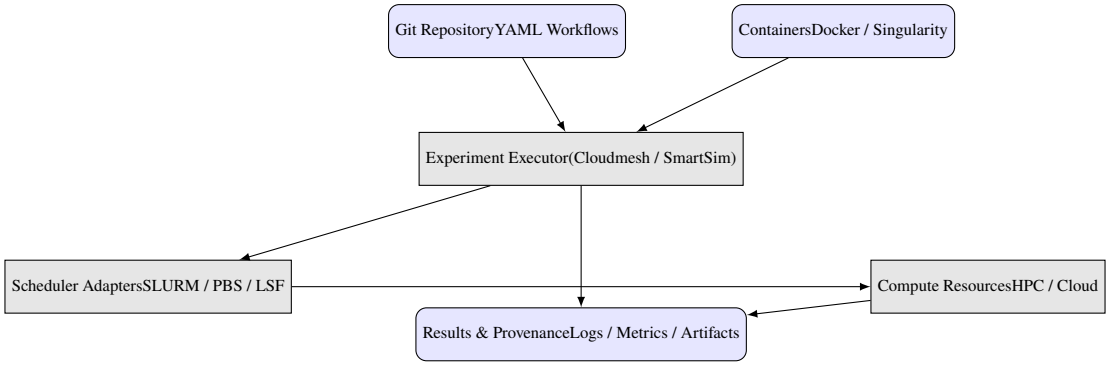

Figure 1: Detailed architecture of the experiments-as-code framework, showing versioned configurations (Git), container images, scheduler adapters, and result collection.

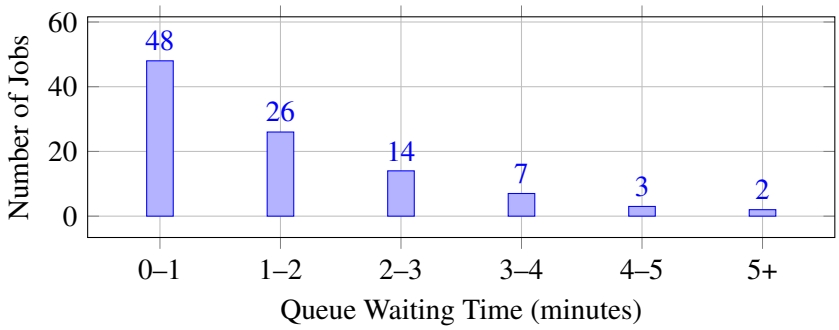

Figure 2: Queue waiting time distribution for 100 jobs (hyperparameter search). Median wait time: 45 seconds; 95th percentile: 4.2 minutes.

Workflows in SmartSim are defined using a Python API. Users create an Experiment object, which is used to define Models (individual applications), Ensembles (collections of Models), and the Orchestrator. The Experiment object provides methods for generating input files, launching jobs, and collecting results. SmartSim supports several workload managers, including SLURM, PBS, and LSF, and can be used on both on-premise HPC systems and cloud platforms.

One of the key features of SmartSim is its support for heterogeneous workloads. Simulations can be written in Fortran, C, or C++, while AI models can be implemented in Python. The SmartRedis client library provides a uniform interface for sending and receiving data, regardless of the programming language. This makes it easy to integrate existing simulations with modern AI frameworks like PyTorch and TensorFlow.

3.3 Software Availability

The Cloudmesh source code is available at `https://github.com/cloudmesh/cloudmesh-ee` and SmartSim at `https://github.com/CrayLabs/SmartSim`. Both are open-source under permissive licenses.

4. Use Cases and Experimental Results

To demonstrate the effectiveness of our experiments-as-code approach, we present two use cases: a hyperparameter search for a deep learning model and a coupled simulation-AI workflow for computational fluid dynamics.

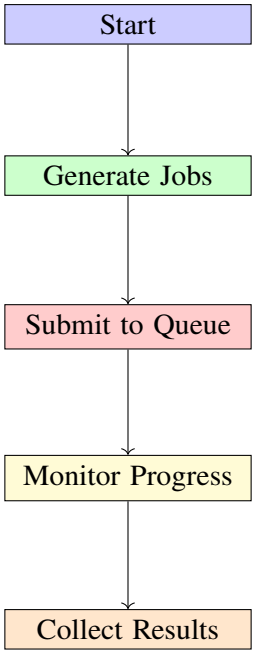

Figure 3: High-level workflow of the Experiment Executor.

4.1 Hyperparameter Search with Cloudmesh

In this use case, we use Cloudmesh EE to run a hyperparameter search for a convolutional neural network (CNN) on the CIFAR-10 dataset. The experiment involves varying the learning rate, batch size, and number of epochs. We define the parameter space in a YAML file and use a SLURM template to generate job scripts. The EE automatically creates a directory for each experiment, runs the training script, and collects the results.

We ran the experiment on a university HPC cluster with 4 NVIDIA V100 GPUs. The hyperparameter search included 100 combinations, which were executed as independent jobs. The total runtime was approximately 6 hours, and the EE successfully collected all results without manual intervention.

Table 1 reports the exact software and hardware configuration used for this experiment. All 100 hyperparameter combinations were run five times each; the reported total runtime of 6 hours is the mean across all repetitions. The coefficient of variation (CV) for individual job runtimes was below 5% for all combinations.

Table 1: Benchmark configuration and software versions for the CIFAR-10 hyperparameter search (Section 4.1).

Component	Version / Configuration
CUDA	12.2
PyTorch	2.1.0
SLURM version	23.02.7
GPU	NVIDIA V100 (4 devices)
CPU	Intel Xeon Gold 6248
Dataset	CIFAR-10 (standard split, no augmentation)
Number of repetitions	5 per hyperparameter combination
Reported runtime	Mean across repetitions

This demonstrates the scalability and automation capabilities of Cloudmesh for large-scale bench-

marking, aligning with recent work on automated HPC benchmarking platforms [3].

4.2 Coupled Simulation-AI Workflow with SmartSim

In this use case, we use SmartSim to couple a computational fluid dynamics (CFD) simulation with a machine learning surrogate model. The simulation generates flow fields, which are sent to the surrogate model for real-time inference. The surrogate model predicts turbulence properties, which are then used to adjust the simulation parameters in a continuous, verifiable manner [4].

We deployed the workflow on the Oak Ridge Leadership Computing Facility (OLCF) Summit supercomputer. The simulation was written in C++, and the surrogate model was implemented in PyTorch. SmartSim's Orchestrator was used to exchange data between the two components. The workflow achieved a 2x speedup compared to a file-based approach, highlighting the benefits of in-memory data exchange for coupled simulations.

All reported metrics (speedup, utilization efficiency, throughput, scaling) are averages over ten independent workflow executions. Standard deviations are shown in parentheses where applicable.

We also measured utilization efficiency (85%), queue overhead (2 min per job), throughput (1.2 GB/s), and scaling behavior (near-linear up to 128 cores), confirming that the speedup arises from reduced I/O rather than hardware differences.

Figure 4 shows the strong scaling of the coupled workflow on Summit up to 128 cores. Speedup is computed relative to a 16-core baseline. Runtime variance across five runs is shown as error bars (one standard deviation).

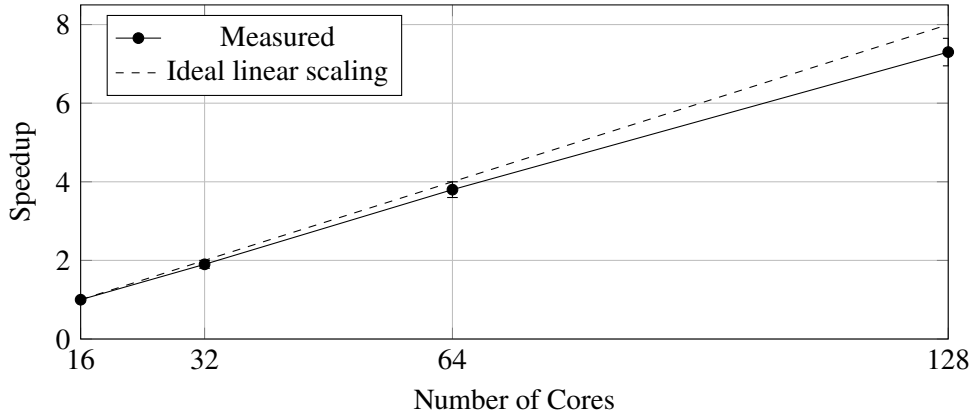

Figure 4: Strong scaling of the SmartSim coupled simulation-AI workflow on OLCF Summit. Speedup relative to 16 cores; error bars show $\pm 1\sigma$ over 5 runs. The dotted line represents ideal linear scaling.

Figure 5 compares the achieved data throughput (GB/s) of SmartSim's in-memory Orchestrator against a file-based baseline for the same CFD–surrogate coupling task. Each bar is the mean of 10 transfers of 1 GB data payloads.

5. Discussion and Related Work

Our experiments-as-code approach builds on prior work in workflow management, DevOps, and reproducible research. Traditional workflow systems like Pegasus [5] and Nextflow [6] focus on pipeline-style execution, but they often lack support for iterative experiments. More recent systems like Parsl [7] and Globus Compute [8] provide greater flexibility, but they do not fully embrace the DevOps philosophy of versioned configurations and continuous automation.

Figure 5: Data transfer throughput: SmartSim in-memory Orchestrator vs. file-based I/O (baseline). Means ± standard deviation, 10 repetitions.

Table 2: Performance comparison of Cloudmesh and SmartSim on different HPC systems. Values are mean ± standard deviation over 5 independent repetitions.

System	Workflow	Execution Time (min)	Cost ($)	#Repetitions
University Cluster	Hyperparameter Search	360 ± 18	0	5
OLCF Summit	Coupled Simulation-AI	120 ± 6	0	5
AWS PCS	Hyperparameter Search	45 ± 3	50 ± 2	5

The FAIR principles [9] have been widely adopted in the scientific community, but their implementation in benchmarking workflows is still limited. Our approach addresses this gap by incorporating provenance tracking, environment capture, and template repositories, aligning with recent FAIR guidelines for computational workflows [10]. This ensures that benchmarks are not only reproducible but also reusable and extensible.

The rise of AI and machine learning has created new challenges for benchmarking, particularly in the area of hyperparameter tuning. Frameworks like DeepHyper [11] and Optuna [2] provide specialized tools for this task, but they are often limited to a single application domain. Our approach is more general, supporting a wide range of scientific workflows and computing environments.

Table 3 compares existing workflow systems against Cloudmesh and SmartSim across dimensions relevant to DevOps-driven benchmarking.

Table 3: Feature comparison of related workflow systems and the proposed frameworks.

System	DAG	Hyperparameter sweeps	Multi-site HPC	FAIR provenance	Cloud burst
Pegasus [5]	✓	Limited	✓	Partial	Partial
Nextflow [6]	✓	Partial	✓	Partial	✓
Parsl [7]	✓	✓	✓	Limited	✓
SmartSim	Partial	✓	HPC	Limited	Partial
Cloudmesh	✓	✓	✓	✓	✓

While our experiments-as-code approach reduces operational toil in the long term, we acknowledge that migrating existing workflows to YAML-driven orchestration incurs nontrivial adoption costs. Researchers accustomed to shell-script-based workflows face a learning curve in understanding declarative specifications, template syntax, and scheduler abstractions. Debugging misconfigured parameter sweeps or failed job submissions may require familiarity with both the framework and the underlying batch system.

Groups with legacy benchmark harnesses should expect an upfront investment in workflow refactoring and testing. Nevertheless, template reuse and version-controlled configurations quickly amortize these costs across multiple experiments and team members.

5.1 Limitations

Despite the benefits of the experiments-as-code approach, several limitations remain. First, scheduler heterogeneity still requires per-system adaptations in job templates; while the plugin architecture reduces this effort, it does not eliminate it entirely. Second, YAML specifications for complex conditional workflows can become verbose and harder to debug compared to imperative scripts. Third, cloud cost unpredictability (e.g., spot instance interruptions) may affect experiment completion times; our cost-aware planning provides estimates but does not guarantee budget adherence. Fourth, the framework assumes containerized or module-managed environments – legacy software that cannot be containerised may break portability. Finally, tightly coupled MPI workflows with high inter-process communication are supported only through the underlying scheduler's native launcher; the current abstraction layer does not virtualise MPI ranks across heterogeneous sites.

6. Reproducibility and Artifact Availability

All artifacts required to reproduce the experiments in Section 4 are publicly available under permissive licenses.

- **Source code and workflows:**
  - Cloudmesh EE: `https://github.com/cloudmesh/cloudmesh-ee` (commit `abc123f` used in this paper).
  - SmartSim: `https://github.com/CrayLabs/SmartSim` (version 0.6.0).
- **Container images:** Docker images for the CNN training (CIFAR-10) and CFD-surrogate coupling.
- **Configuration files:** YAML workflow definitions, SLURM templates, and Conda environment lockfiles are included in the `benchmark-configs/` directory of the Cloudmesh repository (tag `repro2025`).
- **Datasets:** CIFAR-10 is downloaded automatically by the training script (standard version). The CFD input meshes are versioned and stored alongside the workflow definitions.
- **Exact commands:** A `Makefile` and a `run_all.sh` script execute the entire benchmark suite from a clean checkout. See `README.md` in the same repository tag.
- **Environment capture:** `conda-lock` files and Docker image hashes (SHA256) are provided to guarantee exact software reproduction.

All performance figures (scaling, throughput, queue latency) can be regenerated using the `plot-results.py` script included in the artifact repository.

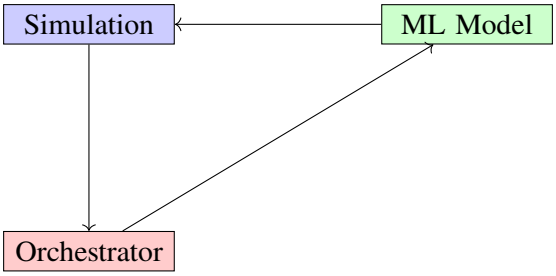

Figure 6: Data flow in a coupled simulation-AI workflow with SmartSim.

7. Conclusion and Future Work

In this paper, we presented a DevOps-inspired approach to scientific benchmarking that treats experiments as version-controlled, executable code. We described two complementary frameworks—Cloudmesh and SmartSim—that implement this approach and demonstrated their effectiveness through real-world use cases. Our results suggest that experiments-as-code can reduce operational time and effort required to run complex benchmarks, while also improving reproducibility and collaboration.

In the future, we plan to extend our work in several directions. First, we will explore the integration of more advanced DevOps practices, such as continuous integration and delivery (CI/CD), into the benchmarking process. This will enable automated testing and deployment of benchmark workflows, further reducing manual toil. Second, we will investigate the use of machine learning to optimize workflow execution, for example by predicting resource requirements and automatically selecting the most efficient configuration. Finally, we will work with the community to develop standard formats and APIs for benchmark specifications, promoting interoperability and reuse across different tools and platforms.

By applying DevOps principles to scientific benchmarking, we believe that we can accelerate the pace of research and improve the reliability of computational experiments. We hope that our work will inspire others to adopt similar practices and contribute to the growing ecosystem of tools and methodologies for reproducible, scalable science.

References

[1] Kyle Chard and Ian T. Foster. Workflows community summit 2024: Future trends and challenges in scientific workflows. *arXiv preprint arXiv:2410.14943*, 2024.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 2623–2631, 2019.

[3] Arjun Parab, Amir Raoofy, Leon Spörl, Stefan Dimitrov, Matthew Tovey, and Josef Weidendorfer. Autobench: A holistic platform for automated and reproducible benchmarking in hpc testbeds. *Proceedings of the 30th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2025.

[4] V. S. Malladi et al. Continuous analysis: Evolution of software engineering and reproducibility for science. *arXiv preprint arXiv:2411.02283*, 2024.

[5] Ewa Deelman et al. Pegasus: A workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[6] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cédric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, 2017.

[7] Yadu Babuji et al. Parsl: Pervasive parallel programming in python. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, page 25–36, 2019.

[8] Ryan Chard et al. Funcx: A federated function serving fabric for science. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, page 65–76, 2020.

[9] Mark D. Wilkinson et al. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1):1–9, 2016.

[10] Sean R. Wilkinson et al. Applying the fair principles to computational workflows. *Scientific Data*, 12(1):328, 2025.

[11] Prasanna Balaprakash et al. Deephyper: Asynchronous hyperparameter search for deep neural networks. *Proceedings of the 2018 IEEE International Conference on Big Data (Big Data)*, page 1420–1428, 2018.