CLOUD COMPUTING 2026

The Seventeenth International Conference on Cloud Computing, GRIDs, and Virtualization

ISBN: 978-1-68558-373-6

April 19 - 23, 2026

Lisbon, Portugal

CLOUD COMPUTING 2026 Editors

Christoph P. Neumann, Ostbayerische Technische Hochschule Amberg-Weiden, Germany
Malte Prieß, Kiel University of Applied Sciences, Germany
Kunal Rao, NEC Laboratories America, Inc., USA
Herodotos Herodotou, Cyprus University of Technology, Cyprus

CLOUD COMPUTING 2026

Forward

The Seventeenth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2026), held on April 19 – 23, 2026, continued a series of events targeted to prospect the applications supported by the new paradigm and validate the techniques and the mechanisms. A complementary target was to identify the open issues and the challenges to fix them, especially on security, privacy, and inter- and intra-clouds protocols.

Cloud computing is a normal evolution of distributed computing combined with Service-oriented architecture, leveraging most of the GRID features and Virtualization merits. The technology foundations for cloud computing led to a new approach of reusing what was achieved in GRID computing with support from virtualization.

The conference had the following tracks:

- Cloud computing
- Computing in virtualization-based environments
- Platforms, infrastructures and applications
- Challenging features
- New Trends
- Grid networks, services and applications

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the CLOUD COMPUTING 2026 technical program committee, as well as the numerous reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to CLOUD COMPUTING 2026. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the CLOUD COMPUTING 2026 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that CLOUD COMPUTING 2026 was a successful international forum for the exchange of ideas and results between academia and industry that will promote further progress in the area of cloud computing, GRIDs and virtualization. We also hope that Lisbon provided a pleasant environment during the conference and everyone saved some time to enjoy this beautiful city.

CLOUD COMPUTING 2026 Steering Committee

Carlos Becker Westphall, Federal University of Santa Catarina, Brazil
Alex Sim, Lawrence Berkeley National Laboratory, USA
Andreas Aßmuth, Kiel University of Applied Sciences, Germany
Aspen Olmsted, Wentworth Institute of Technology, Boston, USA
Christoph P. Neumann, Ostbayerische Technische Hochschule Amberg-Weiden, Germany
Wolfgang Forstmeier Siemens AG, Germany
Malte Prieß, Kiel University of Applied Sciences, Germany

CLOUD COMPUTING 2026 Publicity Chair

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain
Ali Ahmad, Universitat Politècnica de València, Spain
José Miguel Jiménez, Universitat Politècnica de València, Spain
Sandra Viciano Tudela, Universitat Politècnica de València, Spain

CLOUD COMPUTING 2026

Committee

CLOUD COMPUTING 2026 Steering Committee

Carlos Becker Westphall, Federal University of Santa Catarina, Brazil
Alex Sim, Lawrence Berkeley National Laboratory, USA
Andreas Aßmuth, Kiel University of Applied Sciences, Germany
Aspen Olmsted, Wentworth Institute of Technology, Boston, USA
Christoph P. Neumann, Ostbayerische Technische Hochschule Amberg-Weiden, Germany
Wolfgang Forstmeier, Siemens AG, Germany
Malte Prieß, Kiel University of Applied Sciences, Germany

CLOUD COMPUTING 2026 Publicity Chair

Francisco Javier Díaz Blasco, Universitat Politècnica de València, Spain
Ali Ahmad, Universitat Politècnica de València, Spain
José Miguel Jiménez, Universitat Politècnica de València, Spain
Sandra Viciano Tudela, Universitat Politècnica de València, Spain

CLOUD COMPUTING 2026 Technical Program Committee

Sherif Abdelwahed, Virginia Commonwealth University, USA
Vibhatha Abeykoon, Voltron Data Inc., USA
Nikunj Agarwal, Amazon, Inc., USA
Maruf Ahmed, The University of Technology, Sydney, Australia
Mays Al-Naday, University of Essex, UK
Reem Al-Saidi, University of Windsor, Canada
Mubashwir Alam, Marquette University, USA
Abdulelah Alwabel, Prince Sattam Bin Abdulaziz University, Kingdom of Saudi Arabia
Mário Antunes, Polytechnic of Leiria, Portugal
Filipe Araujo, University of Coimbra, Portugal
Mohammad S. Aslanpour, Monash University, Australia
Andreas Aßmuth, Fachhochschule Kiel, Germany
Odiljon Atabaev, Andijan Machine-Building Institute, Uzbekistan
Babak Badnava, University of Kansas, USA
Carlos Jaime Barrios Hernandez, Universidad Industrial de Santander, Colombia
Mohammadreza Barzegaran, University of California Irvine, USA
Luis-Eduardo Bautista-Villalpando, Autonomous University of Aguascalientes, Mexico
Carlos Becker Westphall, Federal University of Santa Catarina, Brazil
Laura Belli, University of Parma, Italy
Leila Ben Ayed, National School of Computer Science | University of Manouba, Tunisia
Nicola Bena, Università degli Studi di Milano, Italy

Salima Benbernou, Universite Paris Cite, France
Simona Bernardi, University of Zaragoza, Spain
Peter Bloodsworth, University of Oxford, UK
Larbi Boubchir, University of Paris 8, France
Jalil Boukhobza, University of Western Brittany, France
Antonio Brogi, University of Pisa, Italy
Roberta Calegari, Alma Mater Studiorum-Università di Bologna,Italy
Jon Calhoun, Clemson University, USA
Juan Vicente Capella Hernández, Universitat Politècnica de València, Spain
Arielle Carr, Lehigh University, USA
Roberto Casadei, Alma Mater Studiorum - Università di Bologna, Italy
Adithya Rajesh Chandrassery, National Institute of Technology Karnataka, Surathkal, India
Ruay-Shiung Chang, National Taipei University of Business, Taipei, Taiwan
Ryan Chard, Argonne National Laboratory, USA
Hao Che, University of Texas at Arlington, USA
Bo Chen, Michigan Technological University, USA
Dawei Chen, InfoTech Labs - Toyota Motor North America R&D, USA
Yitao Chen, Arizona State University, USA
Yue Cheng, George Mason University, USA
Dalila Cherifi, University of Boumerdes, Algeria
Claudio Cicconetti, National Research Council, Italy
Daniel Corujo, Universidade de Aveiro | Instituto de Telecomunicações, Portugal
Fábio M. Costa, Institute of Informatics (INF) | Federal University of Goiás (UFG), Brazil
Alexandre da Silva Veith, Nokia Bell Labs, Belgium
Sajal Dash, Oak Ridge National Laboratory, USA
Luca Davoli, University of Parma, Italy
Patrizio Dazzi, University of Pisa, Italy
Noel De Palma, University Grenoble Alpes, France
Mª del Carmen Carrión Espinosa, University of Castilla-La Mancha, Spain
Simon Pierre Dembele, University of Tartu, Estonia
Frederic Desprez, INRIA, France
Karim Djemame, University of Leeds, UK
Ramon dos Reis Fontes, Federal University of Rio Grande do Norte, Natal, Brazil
Steve Eager, University West of Scotland, UK
Nabil El Ioini, Free University of Bolzano, Italy
Rania Fahim El-Gazzar, Universty of South-Eastern Norway, Norway
Ibrahim El-Shekeil, Metropolitan State University, USA
Levent Ertaul, California State University, East Bay, USA
Javier Fabra, Universidad de Zaragoza, Spain
Fairouz Fakhfakh, University of Sfax, Tunisia
Yuping Fan, Illinois Institute of Technology, USA
Umar Farooq, University of California, Riverside, USA
Tadeu Ferreira Oliveira, Federal Institute of Science Education and Technology of Rio Grande do Norte, Brazil
Sebastian Fischer, University of Applied Sciences OTH Regensburg, Germany
Kaneez Fizza, Swinburne University of Technology, Australia
Stefano Forti, University of Pisa, Italy
Somchart Fugkeaw, Sirindhorn International Institute of Technology | Thammasat University, Thailand

Katja Gilly, Miguel Hernandez University, Spain
Jing Gong, KTH, Sweden
Chander Govindarajan, IBM Research, India
Poonam Goyal, Birla Institute of Technology & Science, Pilani, India
Jordi Guitart, Universitat Politècnica de Catalunya - Barcelona Supercomputing Center, Spain
Saurabh Gupta, Graphic Era Deemed to be University, Dehradun, India
Abdelhay Haqiq, Information Sciences School in Rabat, Morocco
Seif Haridi, KTH/SICS, Sweden
Herodotos Herodotou, Cyprus University of Technology, Cyprus
Uwe Hohenstein, Siemens AG Munich, Germany
Soamar Homsi, Air Force Research Laboratory (AFRL), USA
Md Rajib Hossen, The University of Texas at Arlington, USA
Li-Pang Huang, Tempus, USA
Yujie Hui, Ohio State University, USA
Anca Daniela Ionita, National University of Science and Technology POLITEHNICA Bucharest, Romania
Murat Isik, Stanford University, USA
Mohammad Atiqul Islam, The University of Texas at Arlington, USA
Saba Jamalian, Roosevelt University / Braze, USA
Fuad Jamour, Amazon Web Services (AWS), USA
Weiwei Jia, New Jersey Institute of Technology, USA
Carlos Juiz, University of the Balearic Islands, Spain
Sokratis Katsikas, Norwegian University of Science and Technology, Norway
Zaheer Khan, University of the West of England, Bristol, UK
Ioannis Konstantinou, CSLAB - NTUA, Greece
Sonal Kumari, Samsung R&D Institute, India
Venkatesh Kunchenapalli, Flexport, San Francisco, USA
Rohon Kundu, Lund University, Sweden
Julian Kunkel, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Germany
Giuliano Laccetti, University of Naples Federico II, Italy
Frédéric Le Mouël, INSA Lyon / University of Lyon, France
Kyungyong Lee, Kookmin University, South Korea
Sarah Lehman, Temple University, USA
João Leitão, Universidade Nova de Lisboa, Portugal
Mingchu Li, Jiangxi Normal University, China
Kunal Lillaney, Amazon Web Services, USA
Xue Lin, Northeastern University, USA
Enjie Liu, University of Bedfordshire, UK
Pinglan Liu, Iowa State University, USA
Xiaodong Liu, Edinburgh Napier University, UK
Jay Lofstead, Sandia National Laboratories, USA
Rafael Lopes Gomes, State University of Ceara (UECE), Brazil
Zainab Loukil, University of Gloucestershire, UK
Hui Lu, Binghamton University (State University of New York), USA
Weibin Ma, University of Delaware, USA
Chathura Madhusanka Sarathchandra Magurawalage, InterDigital Europe Ltd., UK
Hosein Mohammadi Makrani, University of California, Davis, USA
Andras Markus, University of Szeged, Hungary
Shaghayegh Mardani, University of California Los Angeles (UCLA), USA

Stefano Mariani, University of Modena and Reggio Emilia, Italy
Attila Csaba Marosi, Institute for Computer Science and Control - Hungarian Academy of Sciences, Hungary
Romolo Marotta, University of l'Aquila (UNIVAQ), Italy
Jean-Marc Menaud, IMT Atlantique, France
Philippe Merle, Inria, France
Nasro Min-Allah, Imam Abdulrahman Bin Faisal University (IAU), KSA
Preeti Mishra, Graphic Era Deemed to be University, Dehradun, India
Takashi Miyamura, NTT Network Service Systems Labs, Japan
Prateeti Mohapatra, IBM Research Lab, India
Francesc D. Muñoz-Escoí, Universitat Politècnica de València, Spain
Ioannis Mytilinis, National Technical University of Athens, Greece
Tamer Nadeem, Virginia Commonwealth University, USA
Hidemoto Nakada, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Akash Nayak, IBM Research, India
Antonio Nehme, Birmingham City University, UK
Richard Neill, RN Technologies LLC, USA
Christoph P. Neumann, Ostbayerische Technische Hochschule Amberg-Weiden, Germany
Bogdan Nicolae, Argonne National Laboratory, USA
Jens Nicolay, Vrije Universiteit Brussel, Belgium
Ridwan Rashid Noel, Texas Lutheran University, USA
Alexander Norta, Tallinn Technology University, Estonia
Aspen Olmsted, Wentworth Institute of Technology, Boston, USA
Matthias Olzmann, noventum consulting GmbH - Münster, Germany
Brajendra Panda, University of Arkansas, USA
Christos Papadopoulos, University of Memphis, USA
Arnab K. Paul, BITS Pilani, India
Sibendu Paul, Amazon Prime Video, USA
Alessandro Pellegrini, National Research Council (CNR), Italy
Sathya Peri, Indian Institute of Technology Hyderabad, India
Nancy Perrot, Orange Innovation, France
Tamas Pflanzner, University of Szeged, Hungary
Paulo Pires, Fluminense Federal University (UFF), Brazil
Agostino Poggi, Università degli Studi di Parma, Italy
Saul E. Pomares Hernandez, Instituto Nacional de Astrofísica, Óptica y Electrónica Tonantzintla, Puebla, Mexico / SARA Group, LAAS-CNRS, Toulouse, France
Pavana Prakash, University of Houston, USA
Walter Priesnitz Filho, Federal University of Santa Maria, Rio Grande do Sul, Brazil
Abena Primo, Huston-Tillotson University, USA
Mohammed A Qadeer, Aligarh Muslim University, India
George Qiao, KLA, USA
Zhihao Qu, Hohai University, China
Francesco Quaglia, University of Rome Tor Vergata, Italy
M. Mustafa Rafique, Rochester Institute of Technology (RIT), USA
Kunal Rao, NEC Laboratories America, USA
Danda B. Rawat, Howard University, USA
Kaustabha Ray, IBM Research, India
Daniel A. Reed, University of Utah, USA

Christoph Reich, Hochschule Furtwangen University, Germany
Sashko Ristov, University of Innsbruck, Austria
Javier Rocher Morant, Universitat Politecnica de Valencia, Spain
Ivan Rodero, Rutgers University, USA
Mohamed Aymen Saied, Laval University, Canada
Benjamin Schwaller, Sandia National Laboratories, USA
Wael Sellami, Higher Institute of Computer Sciences of Mahdia - ReDCAD laboratory, Tunisia
Jayasree Sengupta, Birla Institute of Technology, Mesra, India
Jianchen Shan, Hofstra University, USA
Larisa Shwartz, T.J. Watson Research Center IBM, USA
Muhammad Abu Bakar Siddique, University of California, Riverside, USA
Altino Manuel Silva Sampaio, Escola Superior de Tecnologia e Gestão | Instituto Politécnico do Porto, Portugal
Alex Sim, Lawrence Berkeley National Laboratory, USA
Sima Sinaei, RISE Research Institutes of Sweden, Sweden
Akshit Singhal, University of Texas at Arlington, USA
Bowen Song, University of Southern California, USA
Hui Song, SINTEF, Norway
Polyzois Soumplis, National Technical University of Athens, Greece
Georgios L. Stavrinides, KIOS Research and Innovation Center of Excellence | University of Cyprus, Cyprus
Cesar A. Stuardo, ByteDance, USA
Grażyna Suchacka, University of Opole | Institute of Informatics, Poland
Jingwei Sun, Duke University, USA
Vidhya Suresh, Atlassian Inc , San Francisco, USA
Vasily Tarasov, IBM Research, USA
Zahir Tari, School of Computing Technologies | RMIT University, Australia
Bedir Tekinerdogan, Wageningen University, The Netherlands
Ajay Lotan Thakur, Intel, Canada
Parimala Thulasiraman, University of Manitoba, Canada
Orazio Tomarchio, University of Catania, Italy
Salman Toor, Uppsala University, Sweden
Homero Toral-Cruz, University of Quintana Roo, Mexico
Mert Toslali, IBM Research, USA
Reza Tourani, Saint Louis University, USA
Rajesh Vayyala, PRA Group, Inc., USA
Antonio Virdis, University of Pisa, Italy
Raul Valin Ferreiro, Fujitsu Laboratories of Europe, Spain
Massimo Villari, Università di Messina, Italy
Kewei Wang, Northwestern University, USA
Teng Wang, Oracle, USA
Hironori Washizaki, Waseda University, Japan
Mandy Weißbach, Martin Luther University of Halle-Wittenberg, Germany
Sebastian Werner, Information Systems Engineering (ISE) - TU Berlin, Germany
Michael Wilkins, Northwestern University, USA
Liuqing Yang, Columbia University in the City of New York, USA
Bo Yuan, University of Leicester, UK
Christos Zaroliagis, CTI & University of Patras, Greece

Bo Zhang, Scientific Computing and Imaging Institute | The University of Utah, USA
Zhiming Zhao, University of Amsterdam, Netherlands
Jiang Zhou, Institute of Information Engineering - Chinese Academy of Sciences, China
Yue Zhu, IBM Research, USA
Jan Henrik Ziegeldorf, RWTH Aachen University, Germany
Wolf Zimmermann, Martin Luther University Halle-Wittenberg, Germany

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Rana Nameer Hussain Khan, Dawood Wasif, Jin-Hee Cho, and Ali R Butt

A Modular Kubernetes Workload Simulator for Evaluating Learning-Based Scheduling Policies

Marios Touloupou[1], Syed Mafooq Ul Hassan[1], Jacopo Castellini[2], Pablo Strasser[2], Alexandros Kalousis[2], Herodotos Herodotou[1]
[1]Cyprus University of Technology, Limassol, Cyprus
marios.touloupou@cut.ac.cy, syedmafooqul.shah@cut.ac.cy, herodotos.herodotou@cut.ac.cy
[2]University of Applied Sciences and Arts Western Switzerland (HES-SO), Carouge, Geneva, Switzerland
jacopo.castellini@hesge.ch, pablo.strasser@hesge.ch, alexandros.kalousis@hesge.ch

***Abstract*—Kubernetes is a popular container orchestration platform in cloud and cloud-edge environments. To properly evaluate new Kubernetes scheduling strategies, especially learning-based ones, a controlled but realistic environment is needed, enabling repeatable experimentation without the overhead of running real clusters. This paper presents a modular Kubernetes workload simulator that supports configurable cluster setups, synthetic workload generation, and pluggable scheduling policies. The simulator enables both rule-based and learning-based schedulers to be evaluated under identical conditions, while providing detailed execution traces and performance metrics. This demo paper showcases its capabilities through some interactive scenarios that highlight its usefulness for the development, testing, and comparative evaluation of different Kubernetes schedulers.**

Keywords-Kubernetes; workload simulation; workload scheduling; experimentation.

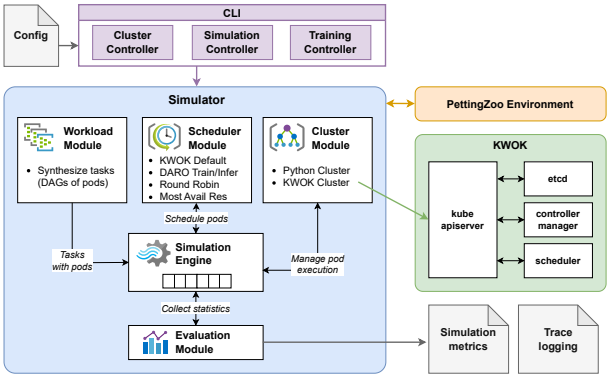


Figure 1. Architecture of the Kubernetes workload simulator.

I. Introduction and Motivation

Evaluating schedulers directly on real cloud or cloud-edge infrastructures is often impractical due to the operational complexity, resource costs, and limited reproducibility of such setups [1]. Moreover, experimentation on real clusters can interfere with production workloads and make it difficult to isolate the impact of scheduling decisions [2].

Recent work highlights the growing interest in learning-based approaches for cloud and cluster resource management, including Kubernetes scheduling [3]. However, existing Kubernetes schedulers and other available simulation environments offer limited support for customized scheduling logic and synthetic workload generation. Many tools either provide fixed scheduling behaviors or operate at a high level of abstraction, which restricts their usefulness for iterative development and fair benchmarking of schedulers [4].

As discussed in recent surveys of cloud, edge, and Internet of Things (IoT) computing, there is an increasing need for lightweight and flexible experimentation environments that bridge the gap between abstract models and real system deployments [5]–[7]. To address these limitations, we designed, implemented, and released a modular open-source Kubernetes workload simulator that provides a controlled, repeatable evaluation framework for different scheduling policies [8]. The simulator was initially motivated by the need to support training and evaluation workflows for learning-based schedulers in a simplified but representative setting. It builds on top of lightweight Kubernetes emulation techniques, such as Kubernetes Without Kubelet (KWOK), to model cluster behavior without the overhead of managing real containers, while extending them with support for dynamic workload synthesis, pluggable scheduling strategies, and detailed execution tracking [9].

The remainder of this paper is structured as follows. Section II presents the architecture and key components of the simulator, Section III demonstrates its use through representative scenarios, and Section IV concludes the paper.

II. Kubernetes Workload Simulator

As depicted in Figure 1, the simulator follows a modular design integrating workload synthesis, scheduling logic, cluster modeling, and evaluation. A command-line interface drives simulation and training workflows, while the simulation engine orchestrates pod execution, scheduling decisions, and statistics collection. The architecture supports interaction with external learning environments and enables both lightweight simulation and Kubernetes-like execution modes.

To balance realism and efficiency, the simulator supports two complementary cluster backends: a Kubernetes-like backend (KWOK [9]) that uses lightweight emulation to model pods and nodes without running actual containers, and a native Python backend with virtual time for fast, controllable experimentation.

The simulator separates cluster modeling, workload generation, scheduling logic, and evaluation into distinct components. The cluster model supports heterogeneous nodes with

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Copyright (c) IARIA, 2026. ISBN: 978-1-68558-373-6

```yaml
# Cluster
cluster_type: Python # KWOK or Python
cluster_reset: True  # True: Clear existing cluster, False: Append to existing cluster

# Cluster Nodes
cluster_nodes_cloud: 20
cluster_nodes_edge: 30
cluster_nodes_iot: 50

# Cloud Node Profile
cluster_node_cloud_cpu_dist: {type: normal, mean: 48000, stdev: 12000, min: 16000, max: 96000, round: -3}  # in millicores
cluster_node_cloud_mem_dist: {type: normal, mean: 96000, stdev: 24000, min: 32000, max: 256000, round: -3} # in Mi
cluster_node_cloud_stg_dist: {type: normal, mean: 4000, stdev: 1000, min: 500, max: 10000, round: -2}      # in Gi
cluster_node_cloud_max_pods: 110

# Edge Node Profile
cluster_node_edge_cpu_dist: {type: normal, mean: 8000, stdev: 2000, min: 4000, max: 16000, round: -2}  # in millicores
cluster_node_edge_mem_dist: {type: normal, mean: 16000, stdev: 4000, min: 4000, max: 32000, round: -2} # in Mi
cluster_node_edge_stg_dist: {type: normal, mean: 500, stdev: 50, min: 50, max: 1000, round: -1}        # in Gi
cluster_node_edge_max_pods: 50

# IoT Node Profile
cluster_node_iot_cpu_dist: {type: normal, mean: 2000, stdev: 500, min: 1000, max: 4000, round: -2} # in millicores
cluster_node_iot_mem_dist: {type: normal, mean: 2000, stdev: 500, min: 1000, max: 4000, round: -2} # in Mi
cluster_node_iot_stg_dist: {type: normal, mean: 16, stdev: 4, min: 1, max: 32, round: 0}           # in Gi
cluster_node_iot_max_pods: 30

# Workload
workload_tasks: 500
workload_pods_number_dist: {type: pareto, alpha: 1.2, min: 2, max: 10, round: 0} # Distribution of pods per task
workload_pods_cpu_dist: {type: normal, mean: 1000, stdev: 500, min: 500, max: 4000, round: -2} # in millicores
workload_pods_mem_dist: {type: normal, mean: 2000, stdev: 500, min: 500, max: 8000, round: -2} # in Mi
workload_pods_stg_dist: {type: normal, mean: 1, stdev: 10, min: 0, max: 500, round: 0}         # in Gi
workload_pods_interarrival_dist: {type: poisson, mean: 20, min: 10, max: 60}                  # in seconds
workload_pods_duration_dist: {type: poisson, mean: 60, min: 10, max: 600}                     # in seconds
workload_pods_max_restarts: 5 # The number of times a pending pod can be restarted (Crashloop error of K8s)

# Scheduler
scheduler_type: ROUNDROBIN  # ROUNDROBIN or DEFAULT or MOSTAVAILABLE or RANDOM or DAROINFER or DAROTRAIN
scheduler_reward_type: CompositeReward  # Coop_LB_reward or Cluster_LB_reward or Node_LB_reward or Fragmentation_reward or
scheduler_composite_reward_parts: [Cluster_LB_reward, Node_LB_reward, Fragmentation_reward] # Rewards for CompositeReward
scheduler_composite_reward_weights: [1, 1, 0] # Reward weights to use with CompositeReward

# Simulation
simulation_speedup: 0  # 1=real-time, 0=infinite, other numbers=speedup factor
simulation_seed: 20    # 0 for fully random, any other number for reproduciable runs
```

Figure 2. Example simulator configuration file used in the demonstration.

configurable CPU, memory, and storage capacities, enabling cloud, edge, and IoT-like environments. Workloads are synthetically generated as sets of pods with configurable resource demands, arrival patterns, and execution durations, ensuring reproducibility.

Scheduling is fully pluggable through a unified interface, allowing different strategies to be evaluated within the same setup. During execution, the simulator coordinates pod arrivals, scheduling decisions, and life cycles using a discrete-event engine. Detailed execution traces and performance metrics are collected at the pod, workload, and cluster levels, enabling analysis of scheduling behavior and resource utilization. These metrics include, among others, pod waiting time, completion time, and resource utilization (CPU, memory, and storage) across nodes.

III. Demonstration Scenarios and Observations

The demonstration illustrates how the Kubernetes Workload Simulator enables the evaluation and comparison of scheduling strategies under controlled conditions. A simulated cluster is configured once, and synthetic workloads are submitted while varying only the scheduling policy. This allows direct comparison of pod placement decisions, resource utilization patterns, and overall cluster behavior using execution traces and performance metrics such as waiting times, completion times, and node-level resource usage.

Initial observations indicate that different scheduling policies yield distinct resource utilization patterns across the cluster, underscoring the need for systematic and repeatable evaluation. Although the current demonstration focuses on CPU, memory, and storage resources, the simulator lays the groundwork for more complex scenarios.

The demonstration is executed through a command-line interface that allows users to configure cluster setups, workloads, and scheduling strategies via configuration files. Users can run simulations, switch between schedulers, and observe their impact on pod placement and resource utilization. The simulator also supports evaluating trained learning-based models and comparing them with baseline approaches, including the default Kubernetes scheduler.

Figure 2 shows an excerpt of the configuration file used to define cluster setups (e.g., cloud, edge, and IoT nodes), workload characteristics, and the scheduling policy. A typical scenario fixes the workload while varying the cluster composition or scheduler to enable consistent comparisons.

IV. Conclusion

This paper presents a modular Kubernetes workload simulator for the controlled, repeatable evaluation of scheduling policies. By combining a Kubernetes-like backend with a native Python backend, it enables flexible experimentation across heterogeneous cloud, edge, and IoT environments. The simulator provides execution traces and performance metrics for analyzing scheduling behavior and resource utilization, supporting iterative development and comparative evaluation. Future work will focus on increasing realism through richer system modeling, tighter integration with real Kubernetes clusters, and the addition of a graphical user interface.

Acknowledgments

This work has been supported by the HYPER-AI project, funded by the European Commission under Grant Agreement 101135982 through the Horizon Europe program.

References

[1] M. Menaka and K. S. Kumar, “Workflow Scheduling in Cloud Environment–Challenges, Tools, Limitations & Methodologies: A Review,” *Measurement: Sensors*, vol. 24, p. 100 436, 2022.

[2] Kubernetes, “Scheduling, Preemption and Eviction,” 2023, [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/ (visited on 02/26/2026).

[3] Q. Zhang, M. Chen, and K. Li, “A Survey on Reinforcement Learning for Cloud Resource Management,” *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–36, 2023. DOI: 10.1145/3566576.

[4] Z. Jian *et al.*, “DRS: A deep reinforcement learning enhanced Kubernetes scheduler for microservice-based system,” *Software: Practice and Experience*, vol. 54, no. 10, pp. 2102–2126, 2024.

[5] S. N. Srirama, “A Decade of Research in Fog Computing: Relevance, Challenges, and Future Directions,” *Software: Practice and Experience*, vol. 54, no. 1, pp. 3–23, 2024.

[6] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D’Andria, “A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues,” *Future Internet*, vol. 15, no. 12, p. 383, 2023.

[7] H. Kuchuk and E. Malokhvii, “Integration of IoT with Cloud, Fog, and Edge Computing: A Review,” *Advanced Information Systems*, vol. 8, no. 2, pp. 65–78, 2024.

[8] Eclipse Research Labs, *Kubernetes Workload Simulator*, 2025. [Online]. Available: https://gitlab.eclipse.org/eclipse-research-labs/hyper-ai-project/k8s-workload-simulator (visited on 02/26/2026).

[9] Kubernetes SIGs, “KWOK: Kubernetes Without Kubelet,” 2023, [Online]. Available: https://kwok.sigs.k8s.io/ (visited on 02/26/2026).

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Platform-Agnostic Resource and Application Profiles for Heterogeneous Cloud–Edge–Device Orchestration

Nektarios Deligiannakis*, Amr Mousa†, Vassilis Papataxiarhis*,
Michalis Loukeris*, Stathes Hadjiefthymiades*
*Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, 15784 Athens, Greece
Email: nekdel@di.uoa.gr, vpap@di.uoa.gr, mloukeris@di.uoa.gr, shadj@di.uoa.gr
†Virtual Vehicle Research GmbH, 8010 Graz, Austria
Email: amr.mousa@v2c2.at

***Abstract*—Orchestration across cloud, edge, and device environments requires consistent and machine-readable representations for both resources and applications. This paper presents a platform-agnostic, schema-validated profiling approach that prioritizes deterministic validation and operational compatibility in control-plane critical paths. The core profile contract is platform-independent, while the current implementation uses Kubernetes primitives where native resource metadata is mapped through labels, annotations, and taints, and externally managed devices are represented through a `DeviceNode` Custom Resource synchronized by controller workflows. For applications, we define versioned profiles for native, device, and integration workloads using JSON Schema Draft-07. We operationalize these models through an Application Profile Manager (APM) that provides profile authoring support, schema validation, repository-backed lifecycle management, and OpenAPI-described programmatic access with ETag-based optimistic concurrency control. Evaluation is functional and correctness-oriented: it validates native/device life-cycle synchronization, schema-enforced profile rejection, repository consistency under concurrent updates, and an end-to-end placement workflow in the Kubernetes-based realization. The result is a practical modeling and governance foundation for continuum orchestration that can be extended to other orchestration substrates through adapter mappings.**

Keywords-Platform-Agnostic Modeling; Kubernetes; Cloud-Edge Continuum; Orchestration; Schema Validation; Application Profiles; Distributed Systems.

I. INTRODUCTION

Automated orchestration across cloud, edge, and device environments requires a common representation for heterogeneous resources and application requirements. In practice, the control plane must combine expressiveness with deterministic validation, interoperability, and manageable lifecycle governance. To address this need, we present a platform-agnostic, schema-validated modeling approach for native nodes, device nodes, and application profiles.

The model is intentionally not tied to a single orchestrator. However, in this paper we implement and evaluate it using Kubernetes primitives because project deployment requirements mandate Kubernetes as the operational baseline [1]. The same schema contract can be mapped to other orchestration platforms through adapter-specific bindings.

The operational component of this approach is the Application Profile Manager (APM), which provides profile authoring support, schema validation, repository persistence, and API-based access. The APM includes three components: a command-line interface (`apmctl`), a versioned schema registry, and a repository service exposed through Representational State Transfer (REST) APIs.

This paper explicitly covers: (i) resource abstraction for native and device representations, (ii) application profile modeling for native/device/integration workloads, and (iii) governance and lifecycle semantics implemented by the APM. We do not claim a novel scheduling or optimization algorithm, and we do not cover compiler/backend workflows beyond profile modeling and management.

Our contributions are:

- **Platform-agnostic resource profile model**: A schema-governed resource description for native and device execution targets, with a Kubernetes mapping based on labels, annotations, taints, and Custom Resource Definitions (CRDs).
- **Device representation and synchronization workflow**: A controller-mediated lifecycle for externally managed devices, realized in this work through `DeviceNode` Custom Resources.
- **Application profiles and governance**: Versioned application profile schemas and an APM toolchain implementing validation, repository consistency controls, and API-based lifecycle management.

The remainder of this paper is structured as follows. Section II presents the background and design rationale. Section III describes the resource and application abstraction layer, covering the native node, device node representation, application profile taxonomy, and the APM. Section IV presents the evaluation, including executed scenarios and an end-to-end orchestration demonstration. Section V discusses the implications and limitations of the proposed approach. Section VI concludes the paper and outlines directions for future work.

II. BACKGROUND AND DESIGN RATIONALE

The design of a scalable continuum orchestration system depends on the representation used for resources and application requirements. A central decision is whether to place semantic reasoning in the control path or to use schema-validated descriptors with explicit attributes and predictable validation behavior.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

A. From Ontologies to Schema-Based Modeling

Ontology-based approaches such as the Web Ontology Language (OWL) provide formal semantics and inference capabilities, but their reasoning overhead and latency variance in tight control loops [2]–[6] motivate our use of schema-based validation instead.

B. Platform-Agnostic, Schema-Validated Models

We adopt a schema-based method using structured JavaScript Object Notation (JSON) [7] and YAML Ain't Markup Language (YAML) [8] documents with versioned validation contracts. The core model is platform-agnostic and can be mapped to different orchestration substrates through adapter layers.

Schema validation provides structural validity and version compatibility guarantees under declared schema versions. In this work, JSON Schema Draft-07 [9] is used as a pragmatic baseline because of mature tooling support and stable validator behavior [9][10]. The model intentionally avoids dynamic-reference-heavy features, prioritizing predictable validation in operational workflows.

This approach also reduces operational complexity compared to architectures that require dedicated semantic infrastructure (e.g., triple stores and reasoners). Profiles are validated with lightweight tools and stored in standard repositories, simplifying integration and maintenance.

C. Resource Models for a Heterogeneous Continuum

A unified model must cover both **native nodes** and **device nodes**. In the Kubernetes realization, native nodes correspond to worker nodes and device nodes are represented in-cluster through CRDs. Kubernetes extensibility mechanisms (labels, annotations, taints, CRDs) provide the required compatibility and incremental extensibility [11].

This representation supports continuum-specific attributes (e.g., accelerator capabilities, network characteristics, trust and energy indicators) while preserving compatibility with existing controllers and schedulers.

D. Application Models for Automated Orchestration

Automated deployment across heterogeneous environments requires machine-readable application profiles describing metadata, resource needs, Quality of Service (QoS) expectations, and deployment constraints. Structured profiles allow orchestration systems to match requirements to capabilities and perform repeatable lifecycle actions.

The APM operationalizes this model by validating profiles against versioned schemas, storing them in a repository, and exposing programmatic APIs.

E. Focused Related Work Positioning

KubeEdge targets device integration at the edge but does not define a portable schema contract for resources and applications across different orchestrators [12]. Node Feature Discovery exposes hardware and software capabilities as labels for scheduling, without a unified profile model [13]. OAM and KubeVela focus on application-centric abstractions for cloud-native workloads and only partially cover continuum-specific attributes such as trust, energy, and mobility [14][15]. Recent systems like QONNECT and REACH concentrate on optimization and control strategies and assume the existence of sufficiently rich descriptors [16][17]. In contrast, this paper contributes a platform-agnostic, schema-governed profiling and governance layer that unifies native and device node models and versioned application profiles, with a concrete but non-binding realization on Kubernetes.

III. Resource and Application Abstraction Layer

This section presents the proposed abstraction layer for resources and applications, covering its design challenges and resource and device representations.

A. Resource Abstraction Layer

The abstraction layer is designed under three main challenges: (i) representing heterogeneous native and device nodes under a single schema, (ii) exposing continuum-specific attributes without introducing ontology reasoning in the control path, and (iii) keeping the model portable while still mapping cleanly to Kubernetes primitives and governance mechanisms.

The resource abstraction distinguishes two categories of computational entities, illustrated in Figure 1:

- **Native Nodes**, corresponding to orchestrator-managed compute targets (implemented here as Kubernetes worker nodes);
- **Device Nodes**, representing computational resources that are externally managed and may not directly join the orchestrator cluster.

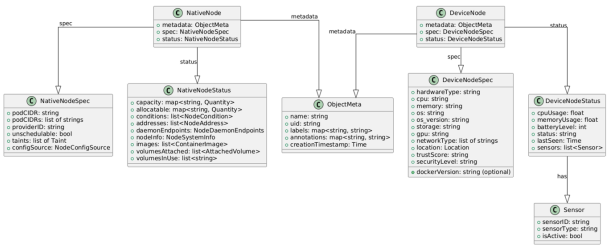


Figure 1. Node types: Native and Device nodes.

This distinction is platform-agnostic at the model level. In this paper, it is bound to Kubernetes-native mechanisms due to project deployment requirements.

B. Kubernetes Realization of the Native Node Model

The standard Kubernetes Node object consists of three primary components:

- `metadata` (ObjectMeta),
- `spec` (NodeSpec),
- `status` (NodeStatus).

These components capture scheduling policies, resource capacity (CPU, memory, storage), and runtime conditions. However, they do not natively represent continuum-relevant attributes such as energy efficiency, monetary cost classes, trust

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE I. EXTENDED NATIVE NODE ATTRIBUTES

Category	Examples
Kubernetes Core	CPU, memory, storage, nodeInfo
Labels	NodeCategory, NodePool, geolocation, accelerator capacity (GPU, FPGA, TPU)
Taints	trustScore, securityLevel
Annotations	energyEfficiency, monetaryCost, bandwidth, latency, packetLoss

levels, geo-location, mobility characteristics, or device-specific metrics.

To address these limitations, the abstraction layer maps the platform-agnostic native node profile to Kubernetes labels, annotations, taints, and related objects. This approach enriches node metadata while remaining fully compatible with Kubernetes APIs and scheduler logic.

Table I summarizes the attribute categories introduced for Native Nodes.

This schema-driven extension preserves low-latency scheduling while enabling richer placement decisions across heterogeneous environments.

In this model, trust and security signals are encoded to support both filtering and exclusion semantics. Labels/annotations expose descriptive attributes for matching and scoring, while taints are used for explicit exclusion or guarded scheduling when policy requires stronger protection boundaries.

C. Device Node Representation via CRDs

Device nodes do not operate as orchestrator workers. In the Kubernetes implementation, they are represented within the control plane as Custom Resources following the standard `metadata/spec/status` structure.

Each `DeviceNode` resource captures static attributes (e.g., hardware type, CPU/GPU model, operating system) and dynamic runtime metrics (e.g., CPU utilization, memory usage, connectivity status, battery level). This design ensures that orchestration components can query and monitor device capabilities through native control-plane APIs.

The system uses controller-based methods to keep physical devices in sync with their associated Custom Resources. Devices send their registration and status information through Mosquitto Message Queuing Telemetry Transport (MQTT) [18]. The Kubernetes API server creates or updates a Custom Resource after the DeviceNode schema validation process.

If expected updates are missed beyond configured timeout bounds, the controller marks the device offline and temporarily removes it from the scheduling candidate pool. This reconciliation mechanism leverages Kubernetes' controller pattern [19] to maintain consistency between desired and observed state.

D. Design Properties

The Node Model abstraction satisfies the following properties:

- **Portability**: Platform-agnostic core schema with adapter-based platform mappings;
- **Compatibility**: Kubernetes-compatible realization using native primitives;
- **Extensibility**: Incremental addition of new attributes without schema disruption;
- **Determinism**: Schema-based validation avoids runtime reasoning overhead;
- **Scalability**: Event-driven updates support large fleets of heterogeneous nodes;
- **Low Latency**: No ontology reasoning in the scheduling critical path.

By modeling both Native and Device nodes through structured, schema-validated representations, the abstraction layer enables uniform resource discovery, monitoring, and scheduling across the computing continuum.

E. Application Profile Model

An Application Profile (AP) is a declarative YAML document that describes an application's operational requirements and intended deployment state. The model is designed to provide scalable, reproducible, and schema-validated application descriptions suitable for automated orchestration across heterogeneous environments.

Each profile follows a platform-neutral `metadata/spec/status` structure (with Kubernetes-inspired naming in our implementation):

- **metadata**: Identifying information such as name, namespace, labels, and annotations.
- **spec**: Desired state definition, including resource requirements, dependencies, Quality of Service (QoS) parameters, and deployment constraints.
- **status**: Observed runtime state, maintained by orchestration.

Profiles are versioned and validated against JSON Schema Draft-07 specifications, ensuring structural consistency and forward compatibility.

F. Profile Taxonomy

Three profile categories enable unified modeling across execution domains:

- **Native Profiles**: Target containerized or virtualized workloads deployed on orchestrator-managed native compute nodes (Kubernetes workers in this implementation).
- **Device Profiles**: Describe applications interacting with hardware-specific resources in edge and Internet of Things (IoT) environments.
- **Integration Profiles**: Define multi-component or composite services, including API endpoints, external dependencies, and orchestration policies.

Each category extends a shared base schema with type-specific attributes, ensuring modularity while preserving structural uniformity. Figures 2, 3 and 4 depict the three above profiles.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

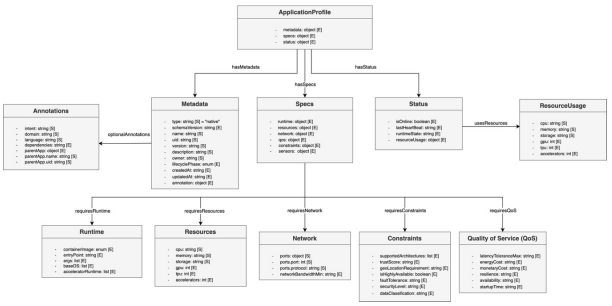


Figure 2. Data model diagram for a native application profile.

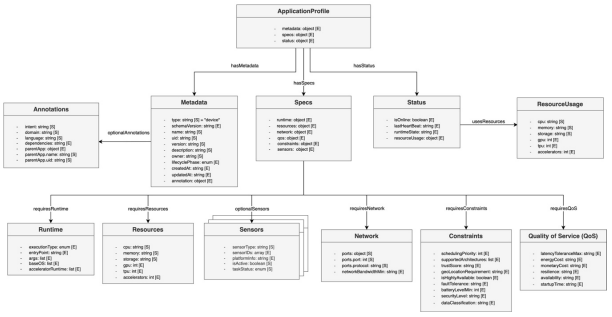


Figure 3. Data model diagram for a device application profile.

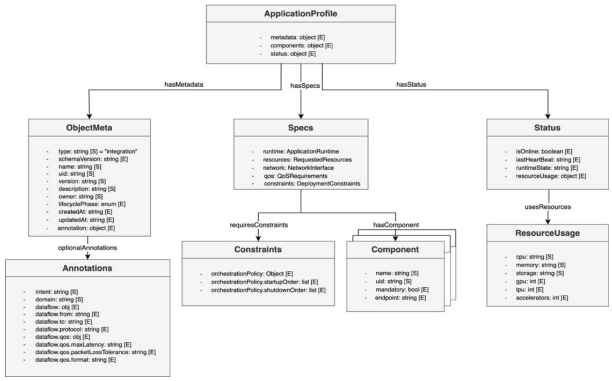


Figure 4. Data model diagram for an integration application profile.

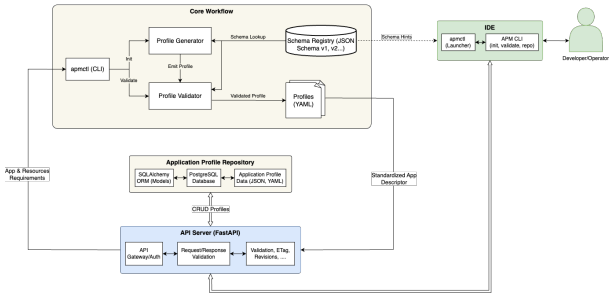


Figure 5. High-level architecture of the Application Profile Manager.

G. Schema Modularity and Attribute Scope

The schema design is modular and layered. A base schema defines common constructs (metadata, status, core identifiers), while type-specific schemas extend it with domain-relevant attributes.

The logical structure is organized into three primary components:

- **Core Definition**: Metadata, execution context, and QoS requirements.
- **Resource Requirements**: CPU, memory, accelerators, storage, and network specifications.
- **Deployment Constraints**: Scheduling preferences, geographical restrictions, trust parameters, and platform-specific requirements.

To clarify attribute origin, fields are categorized as:

- **Core**: Platform-agnostic attributes used across orchestrators;
- **Adapter-Specific**: Attributes mapped to a concrete platform realization (e.g., Kubernetes labels/taints/CRDs).

This separation preserves portability while enabling direct interoperability with Kubernetes-native tooling in the current implementation.

H. Application Profile Manager (APM)

The APM provides lifecycle management for application profiles, including authoring, validation, storage, and programmatic access, as illustrated in Figure 5.

The APM consists of three primary components:

- **Command-Line Interface (CLI) (`apmctl`)**: Enables template generation, schema validation, and repository interaction.
- **Schema Registry**: A versioned repository of JSON Schemas and corresponding YAML templates.
- **Profile Repository**: A persistent service exposing RESTful APIs for Create, Read, Update, Delete (CRUD) operations on profiles.

Validation is enforced using JSON Schema Draft-07 and automated through the `check-jsonschema` tooling. During validation, YAML profiles are converted to JSON and checked for required fields, data types, patterns, and enumerations according to the declared profile type and schema version.

The repository API is implemented using FastAPI [20] and automatically exports an OpenAPI 3 specification [21]. This provides synchronized interface documentation and standards-compliant programmatic access. The governance contract includes: (i) schema-version binding for each stored profile, (ii) explicit validation failure reporting for structural/type/constraint violations, and (iii) HTTP ETag-based optimistic concurrency control to prevent lost updates during concurrent writes.

The Application Abstraction Layer provides:

- **Deterministic Validation**: Schema-based enforcement eliminates runtime ambiguity;
- **Modularity**: Layered schema extensions support heterogeneous execution targets;
- **Interoperability**: Platform-agnostic schema contracts with a Kubernetes reference mapping for implementation compatibility;
- **Lifecycle Management**: Versioned profiles enable reproducible deployments and controlled evolution.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

By combining structured application descriptors with a schema-governed tool-chain, the abstraction layer enables automated matching between application requirements and resource capabilities across the computing continuum, while allowing extension to non-Kubernetes control environments through adapter mappings.

IV. EVALUATION

The proposed modeling layer was validated in heterogeneous environments, including local clusters, on-premises multi-node setups, and cloud-based infrastructure. The reported experiments correspond to the Kubernetes-based implementation selected for the project environment. The evaluation is functional and correctness-oriented: it verifies lifecycle synchronization, validation behavior, and repository consistency under representative scenarios.

A. *RQ1: Does native node metadata synchronization remain correct under lifecycle events?*

Native nodes were instrumented through a DaemonSet-based synchronization loop. In each cycle, static attributes were read from the Kubernetes API, while dynamic metrics (e.g., network indicators, uptime, energy-related attributes) were recomputed and patched to the Node object using JSON patch operations.

Observed outcomes:

- Enriched metadata propagated to active worker nodes.
- Node join and node removal events were reflected without manual intervention.
- Dynamic attribute refresh did not disrupt scheduler operation during the test scenarios.
- When a metric source was unavailable, fallback behavior preserved profile consistency.

B. *RQ2: Is device node registration and status reconciliation reliable?*

Device nodes were validated using an MQTT-based registration and synchronization workflow. Registration payloads were schema-checked before CR creation. Runtime metrics were patched to the `status` field through controller reconciliation.

Observed outcomes:

- Registration events produced valid `DeviceNode` resources with expected metadata/spec/status structure.
- Policy-based availability checks (e.g., battery-related constraints) were reflected in scheduling eligibility.
- Timeout-based liveness handling marked stale devices offline and removed them from candidate pools.
- Dynamic capability changes were propagated to the corresponding CR status.

Controller logs during the executed scenarios show successful API patch operations and consistent reconciliation behavior in our testbed.

C. *RQ3: Do schema validation and repository controls enforce profile consistency?*

Application profiles were validated against JSON Schema Draft-07. Invalid inputs were rejected for structural violations, type mismatches, and enumeration errors. Versioned schemas constrained accepted structures by declared profile type/version.

The repository API enforced optimistic concurrency using HTTP ETags. Concurrent modification scenarios showed stale updates being rejected with precondition failures, preventing lost updates.

D. *Executed scenarios and evidence*

TABLE II. EXECUTED SCENARIOS AND OBSERVED OUTCOMES

Scenario	Expected	Observed	Evidence
Native metadata refresh	Node metadata enriched and updated after lifecycle events	Successful propagation and refresh in test runs	DaemonSet/controller logs and Node object state snapshots
Device registration	Valid registration creates/updates `DeviceNode` CR	CR created with expected fields	MQTT ingestion logs, controller events, CR snapshots
Device liveness timeout	Stale device marked unavailable/offline	Candidate removed after timeout handling	Controller logs and CR status transitions
Profile validation	Invalid profile rejected before persistence	Invalid inputs rejected by schema checks	Validation CLI/API responses
Concurrent profile updates	Stale write rejected to avoid lost update	Outdated ETag rejected (precondition failure)	Repository API request/response logs

E. End-to-end orchestration demonstration

An end-to-end scenario was executed in which an application profile specifying hardware type, operating system, battery threshold, and sensor requirements was submitted to the system. The scheduler successfully matched the application to a compatible Device Node and delegated execution. Upon device state changes (battery drop or network loss), the system automatically excluded the node and re-evaluated placement decisions.

These results show that the proposed layer provides deterministic schema validation, stable synchronization semantics, and repository consistency mechanisms suitable for continuum orchestration workflows. Large-scale throughput, tail-latency, and comparative baseline benchmarking are left for future work.

V. DISCUSSION

The proposed modeling layer represents a deliberate shift from ontology-centric runtime reasoning toward schema-validated representations. This decision reflects a trade-off between formal semantic expressiveness and operational predictability in control-plane workflows.

Ontology-driven approaches provide strong reasoning capabilities but can introduce additional overhead and latency variance as models and engines grow more complex. In distributed

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

environments with tight feedback loops, this variance is problematic, whereas schema-based validation performs structural checks before runtime placement, keeping the critical path simpler and more predictable.

Nevertheless, schema-based modeling imposes certain limitations. Unlike ontology-driven systems, it does not support automated semantic inference beyond explicitly defined attributes. Complex reasoning tasks must therefore be implemented in higher-level optimization components rather than embedded within the data representation layer. The architecture intentionally separates descriptive modeling from optimization logic to preserve performance and maintainability.

Overall, the abstraction layer demonstrates that structured, schema-governed models can provide sufficient expressiveness for continuum orchestration while maintaining deterministic behavior and scalability at runtime.

VI. Conclusions and Future Work

This paper presented a platform-agnostic, schema-governed resource and application profiling layer for cloud–edge–device orchestration. The approach combines platform-neutral profile contracts with a Kubernetes realization (native node enrichment and `DeviceNode` CRDs), together with versioned application profiles and an APM governance toolchain.

The evaluation focused on functional correctness in the Kubernetes-based implementation: synchronization behavior, schema validation outcomes, repository consistency, and end-to-end orchestration flow under representative scenarios. These results support the claim that the proposed layer is a practical and deterministic foundation for higher-level continuum orchestration components.

Future work will focus on three directions. First, quantitative scale and performance studies will measure control-plane overhead, update throughput, and tail latencies under stress. Second, security hardening will address identity, authorization, admission control, and multi-tenant boundaries. Third, integration with optimization/scheduling frameworks will evaluate how enriched descriptors affect placement quality and Service Level Objective (SLO) outcomes.

Acknowledgments

This work has been partially supported by the HYPER-AI project, funded by the European Commission under Grant Agreement 101135982 through the Horizon Europe research and innovation program (https://hyper-ai-project.eu/).

References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016. DOI: 10.1145/2890784.

[2] V. Sazonau, U. Sattler, and G. Brown, "Predicting performance of OWL reasoners: Locally or globally?" In *Proceedings of the International Semantic Web Conference (ISWC)*, 2012, pp. 1–16.

[3] W. V. Woensel and S. S. R. Abidi, "Optimizing and benchmarking OWL2 RL for semantic reasoning on mobile platforms," *Journal of Web Semantics*, vol. 27–28, pp. 1–18, 2013.

[4] C. Seitz and R. Schönfelder, "Rule-based OWL reasoning for specific embedded devices," in *Proceedings of the International Semantic Web Conference (ISWC)*, 2011, pp. 237–252. DOI: 10.1007/978-3-642-25093-4_16.

[5] W. Tai, J. Keeney, and D. O'Sullivan, "COROR: A composable rule-entailment OWL reasoner for resource-constrained devices," in *Proceedings of the International Web Rule Symposium (RuleML)*, 2011, pp. 212–226. DOI: 10.1007/978-3-642-22546-8_17.

[6] A. I. Maarala, X. Su, and J. Riekki, "Semantic reasoning for context-aware internet of things applications," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 461–473, 2017. DOI: 10.1109/JIOT.2016.2587060.

[7] T. Bray, *The JavaScript Object Notation (JSON) Data Interchange Format*, RFC 8259, [retrieved: April, 2026], 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8259.

[8] *YAML Ain't Markup Language (YAML) Version 1.2*, [retrieved: April, 2026], 2009. [Online]. Available: https://yaml.org/spec/1.2/spec.html.

[9] "JSON Schema Draft-07," [retrieved: April, 2026], 2025. [Online]. Available: https://json-schema.org/draft-07.

[10] "Check-jsonschema," [retrieved: April, 2026], 2025. [Online]. Available: https://github.com/python-jsonschema/check-jsonschema.

[11] The Kubernetes Authors, "Custom resource definitions," [retrieved: April, 2026], 2025. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[12] KubeEdge Authors, "Kubeedge documentation," [retrieved: April, 2026], 2026. [Online]. Available: https://kubeedge.io/docs/.

[13] Kubernetes SIG Node, "Node feature discovery," [retrieved: April, 2026], 2026. [Online]. Available: https://kubernetes-sigs.github.io/node-feature-discovery/.

[14] Open Application Model Community, "Open application model (oam) specification," [retrieved: April, 2026], 2026. [Online]. Available: https://oam.dev/.

[15] KubeVela Authors, "Kubevela documentation," [retrieved: April, 2026], 2026. [Online]. Available: https://kubevela.io/docs/.

[16] QONNECT Authors, "QONNECT: QoS-aware continuum orchestration with kubernetes-native control planes," [retrieved: April, 2026], 2025. [Online]. Available: https://arxiv.org/abs/2510.09851.

[17] REACH Authors, "REACH: Reinforcement learning-based rescheduling in heterogeneous kubernetes continuum environments," [retrieved: April, 2026], 2025. [Online]. Available: https://arxiv.org/abs/2510.06675.

[18] *MQTT version 3.1.1*, [retrieved: April, 2026], OASIS, 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/.

[19] The Kubernetes Authors, "Kubernetes controllers," [retrieved: April, 2026], 2025. [Online]. Available: https://kubernetes.io/docs/concepts/architecture/controller/.

[20] "Fastapi framework," [retrieved: April, 2026], 2025. [Online]. Available: https://fastapi.tiangolo.com/.

[21] "Openapi specification," [retrieved: April, 2026], 2025. [Online]. Available: https://github.com/OAI/OpenAPI-Specification.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Hypertool: A Resource Abstraction and Lifecycle Management Framework for Heterogeneous Cloud Environments

Michalis Loukeris[1], Nektarios Deligiannakis[1], Vassilis Papataxiarhis[1], Panagiotis Kechriniotis[1], Syed Mafooq Ul Hassan[2], Nicolas Louca[3], Herodotos Herodotou[2], Stathes Hadjiefthymiades[1]
[1] Department of Informatics and Telecommunications
National and Kapodistrian University of Athens, 15784 Athens, Greece
e-mail: {mloukeris, nekdel, vpap, pkehri, shadj}@di.uoa.gr
[2] Department of Electrical Engineering, Computer Science and Engineering
Cyprus University of Technology, 3036 Limassol, Cyprus
e-mail: {syedmafooqul.shah, herodotos.herodotou}@cut.ac.cy
[3] eBOS Technologies Ltd
2043 Nicosia, Cyprus
e-mail: nicolasl@ebos.com.cy

***Abstract*—Orchestrating modern applications across distributed cloud environments requires a unified, platform-agnostic representation of heterogeneous computational resources and a tool to automate their management. This paper presents an open-source framework, Hypertool, designed to automate node registration, discovery, and lifecycle management. Additionally, it provides an extensible model that enhances the default node representation with additional attributes, such as hardware acceleration, energy efficiency, and security posture, which are vital for next-generation workloads. By implementing a dual Command Line Interface (CLI) and DaemonSet architecture, the framework acts as an all-in-one utility for orchestrating complex functionalities, abstracting computing, networking, and economic metrics into declarative data models. The current implementation leverages core Kubernetes primitives, ensuring effortless integration into existing heterogeneous cloud environments. The framework is evaluated across local, on-premises, and public cloud environments, demonstrating high operational maturity, stability, and production-readiness.**

Keywords-Resource Abstraction; Distributed Cloud Environments; Kubernetes; Node Lifecycle Management; Heterogeneous Computing; Declarative Data Models; Orchestration.

I. INTRODUCTION

Orchestrating modern workloads across distributed cloud environments presents a twofold challenge. First, advanced cloud-edge applications, particularly those involving IoT, AI, and latency-sensitive tasks, require a more sophisticated representation of computational resources than the defaults provided by standard orchestration platforms such as Kubernetes [1][2]. To facilitate intelligent workload placement, systems require an extended node schema that captures dynamic attributes, such as energy efficiency, hardware acceleration, and security posture, across the cloud-to-edge continuum [3][4]. Second, the lifecycle management of these distributed resources remains a significant hurdle. As heterogeneous nodes are added to or removed from a cluster, existing manual procedures are often complex, multi-step, and error-prone [5].

To address these limitations, this paper introduces **Hypertool**, an open-source framework that automates the registration, discovery, and lifecycle management of nodes in heterogeneous cloud architectures. Hypertool provides a unified, platform-agnostic resource abstraction layer that simplifies the onboarding of diverse assets while alleviating vendor lock-in. It is also engineered for seamless deployment on any Kubernetes-compatible infrastructure, ensuring interoperability across local, on-premises, and public cloud environments.

The framework employs a dual-component architecture consisting of a cross-platform **CLI** and a persistent **DaemonSet**[6]. Within the Kubernetes ecosystem, a DaemonSet is a controller that ensures a copy of a specified pod runs on every node in the cluster, providing a resilient mechanism for background services. The CLI offers an intuitive interface for administrators to handle node onboarding, including a "dry run" mode to simulate operations before execution. Simultaneously, the daemon service runs on every node to continuously monitor health, recalculate resource attributes, and update the Kubernetes control plane via declarative Custom Resource Definitions (CRDs) [7].

Hypertool is built with a secure-by-design philosophy and adheres to industry best practices for authentication and authorization. To ensure high operational maturity, the project maintains comprehensive Continuous Integration/Continuous Deployment (CI/CD) pipelines for continuous testing and delivery, supported by extensive documentation and deployment guides.

The primary contributions of this work are as follows:

- **Resource Abstraction Layer**: A declarative data model that abstracts resources across the continuum, continuously calculating dynamic attributes to provide an enhanced representation of heterogeneous computational power.
- **Automated Lifecycle Management**: A robust mechanism to manage the complete lifecycle of distributed resources, significantly reducing the complexity of node registration and cluster integration.
- **Production-Ready Framework**: An open-source, secure-by-design utility featuring a dual CLI/DaemonSet architecture, validated through automated CI/CD

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

pipelines and evaluated across diverse cloud environments for stability and performance.

The remainder of this paper is organized as follows: Section II reviews related work. Section III details the Hypertool framework architecture. Section IV presents the system's validation across diverse environments, and Section V concludes the paper.

II. Related Work

Existing operational frameworks and modern orchestration platforms, such as Kubernetes [1], OpenStack [8], and HashiCorp Nomad [9], serve as the industry standard for deploying distributed applications. However, while these systems provide a generic model for representing compute resources, their default models are fundamentally basic, typically limited to static allocations of CPU cores, memory, and storage capacity. Although extensibility mechanisms exist to enhance these representations (such as Custom Resource Definitions in Kubernetes or custom metadata flavors in OpenStack), the baseline schema remains too simplistic [7][10]. It cannot meet the multidimensional needs of modern applications, which increasingly require real-time awareness of complex attributes such as energy efficiency, specialized hardware acceleration, and network performance metrics to make optimal scheduling decisions.

Furthermore, current procedures for managing node lifecycles within these platforms inherently assume that the underlying infrastructure is highly stable and static. When heterogeneous resources need to be dynamically added to or removed from a cluster, the default lifecycle management processes prove inadequate [2][5]. The built-in mechanisms for registering, updating, or deleting nodes typically involve multiple disjoint, intricate steps. Consequently, these operations are highly complex, error-prone, and demand significant domain expertise, often requiring a certified cluster administrator to manually execute and verify the changes safely. This rigidity prevents automated scaling in modern distributed cloud architectures, underscoring the critical need for a more dynamic, automated approach to resource abstraction and lifecycle governance.

III. Hypertool Framework Architecture

A. Hypertool as a CLI Tool

To effectively resolve the twofold challenge of providing a sophisticated resource representation and automating dynamic node lifecycle management, the framework introduces a dual-component architecture comprising a Command Line Interface (CLI) utility and a management DaemonSet, as depicted in Figure 1. This structural separation of concerns ensures that heterogeneous computing resources are first thoroughly profiled and abstracted into declarative data models prior to cluster inclusion, and subsequently continuously monitored, updated, and managed during their operational lifecycle.

The CLI utility serves as the primary interface for cluster administrators to govern the end-to-end lifecycle of computational nodes, as illustrated in Figure 2. It is specifically designed to manage the critical phases of resource onboarding and offboarding, including initial discovery, self-advertisement, node registration, and unregistration.

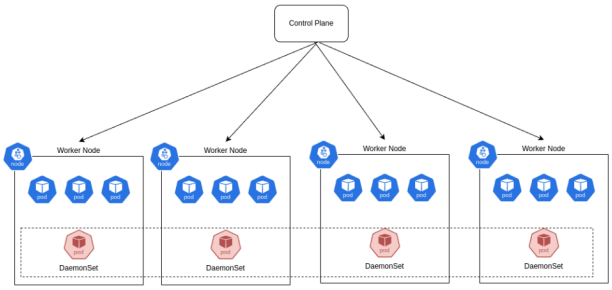


Figure 1. The dual-component architecture of the Hypertool framework, showing the interaction between the CLI profiler and the management DaemonSet.

```
> hypertool run --dry-run
IP address of the master node: 8.8.8.8

[DRY-RUN] The following attributes would be applied:

[+] GEOLOCATION
    City:    Athens
    Region:  Attica
    Country: GR

[+] PERFORMANCE & COST
    Cost Category:     Very Low
    Energy Efficiency: Very Low
    FLOPS/sec:         11808288288.29
    Latency Class:     A

[+] NETWORK
    Network Type: Ethernet
    Interface:    enp6s0
```

Figure 2. Overview of the CLI utility and node lifecycle management.

By encapsulating the underlying complexity of the native orchestration platform, the application translates intricate, multi-step Kubernetes API [1] interactions into a centralized suite of intuitive commands (e.g., `register`, `unregister`, `self-advertisement`). This design significantly reduces the operational overhead and domain expertise traditionally required to manage heterogeneous clusters. Furthermore, while the default execution abstracts these complex mechanisms to provide a streamlined user experience, the tool includes a configurable verbose mode. When enabled, this mode surfaces granular, low-level diagnostic logs and API responses, equipping administrators with deep visibility for advanced troubleshooting.

Crucially, the CLI component is engineered with sophisticated error management and fault tolerance to ensure operational continuity. Recognizing the inherent unpredictability of distributed infrastructures, the tool employs robust exception handling and automatic rollback mechanisms to prevent catastrophic failures. Rather than experiencing a total system crash when an error occurs during command execution, the application logs an informational warning and automatically reverts the system to its previous stable state.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

B. Hypertool as a DaemonSet

While the CLI utility handles the initial registration, the nodes' continuous lifecycle management is maintained by deploying Hypertool as a Kubernetes DaemonSet [6]. By utilizing this native orchestration resource, the framework ensures that a dedicated monitoring pod is automatically scheduled and deployed to every new worker node that joins the cluster. This approach leverages Kubernetes's inherent self-healing and reconciliation loops, ensuring that the Hypertool remains operational and resilient against localized failures.

To ensure that resource information remains up to date and accurately reflects a node's current status, Hypertool runs a continuous polling routine. By default, this execution loop runs every five minutes, strictly aligning with the native `-node-status-update-frequency` of the Kubelet mechanism [11]. During each interval, the pod dynamically recalculates an extensive suite of system attributes and patches the orchestration API, transitioning the node representation from a static asset to a highly dynamic, robustly profiled entity. The primary metrics dynamically calculated during this phase include:

- **Computational Performance**: The attribute is computed by first executing a quick subsecond artificial workload to estimate floating point operations per second, and then computing GFLOPs (Giga Floating-Point Operations per Second). It serves as a hardware performance indicator derived from synthetic workloads and is crucial for benchmarking, load balancing, and compute-aware scheduling decisions.
- **Energy Efficiency**: This attribute quantifies performance-per-watt (GFLOPs/Joule). It is calculated by correlating the dynamically estimated FLOPs with the node's Thermal Design Power (TDP). The TDP is either retrieved via hardware introspection databases or predicted using a trained Random Forest regressor [12] based on core count, CPU clock speed (GHz), and L3 cache size from Intel- and AMD-provided datasets.
- **Monetary Cost**: To enable cost-aware scheduling, a pre-trained K-Means clustering [13] model evaluates the instance's CPU and memory specifications. It maps the node's Euclidean distance against curated cloud pricing centroids to assign a qualitative monetary tier (ranging from *very low* to *very high*) without requiring real-time pricing APIs.
- **Trust Score**: This metric provides a quantified assessment of a node's operational stability. It computes a weighted score by aggregating internal stability signals, such as container restart frequencies, CPU throttling periods, pod evictions, and Out-Of-Memory (OOM) kills, classifying the node's reliability as high, medium, or low.
- **Security Posture**: To safeguard sensitive workloads, the framework calculates a composite security level. This weighted metric evaluates system patch compliance, Center for Internet Security (CIS) benchmark passing rates [14], open CVEs [15] detected via rootfs scanning, and overly permissive RBAC rules.
- **Network Performance & Geolocation**: The framework periodically executes active networking tests to quantify download/upload bandwidth, Round-Trip Time (RTT), and packet loss percentage. Additionally, it approximates the node's geographic location (city, region, country) utilizing IP geolocation mappings to facilitate edge-aware workload placement.
- **Hardware Accelerators**: It dynamically scans and exposes the exact capacity of specialized hardware natively available on the host, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs), enabling schedulers to target nodes specifically suited for accelerated machine learning tasks.
- **Node Category**: This attribute provides a concatenated architectural summary of the node's hardware profile. Following a lightweight system scan, the framework generates an acronym representing the node's primary capabilities, specifically evaluating CPU core count, RAM capacity, disk size, and the presence of hardware accelerators.
- **Node Pool**: Inspired by major cloud provider architectures, this attribute enables the logical grouping of nodes based on their hardware capabilities, functional purpose, or both. It serves as an administrative abstraction layer for organized cluster management.
- **Node Uptime**: This attribute captures the operational lifespan of the node, serving as a temporal fingerprint within large-scale, dynamic clusters where nodes are frequently decommissioned and replaced. Given its uniqueness in volatile environments, the metric can serve as a supplementary identifier for the node.

IV. VALIDATION

To ensure operational maturity, robustness, and ease of adoption, the proposed framework was validated across multiple heterogeneous environments. Hypertool is officially packaged and published as a Helm chart [16]. This standardized packaging guarantees that the framework can be effortlessly integrated, updated, and maintained within any compliant Kubernetes cluster with minimal administrative overhead. By following this cloud-native deployment strategy, the framework ensures a validated and repeatable installation process regardless of the underlying infrastructure.

The testing methodology was divided into three distinct operational phases to evaluate different architectural requirements:

- **Local Validation**: This phase utilized Minikube [17] to validate the primary features of Hypertool in a lightweight, single-node Kubernetes setting. The machine used for these tests was a 12-core system with 16 GB of RAM and a 250 GB SSD. Hypertool successfully registered the single node and correctly computed all resource metrics listed in Section III.B.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

- **On-Premises Validation**: To evaluate the framework under standard multi-node conditions, an on-premises virtualized cluster was provisioned. This cluster consisted of a dedicated control plane (4 vCPUs, 6 GiB RAM, 64 GiB of HDD storage) and multiple worker nodes (3 vCPUs, 4 GiB RAM, 32 GiB HDD storage), enabling rigorous testing of resource isolation, inter-node coordination, and the DaemonSet's ability to seamlessly scale and attach to newly joined nodes.
- **Public Cloud Validation**: The final validation phase was executed on public cloud infrastructure to simulate a highly heterogeneous, production-grade continuum. The cluster incorporated a diverse mix of compute resources, including general-purpose instances (4–16 vCPUs, 8–64 GiB RAM), GPU-optimized hardware, and more. This phase successfully demonstrated the framework's capability to accurately profile varying raw compute capacities, detect specialized hardware accelerators, and dynamically estimate cost tiers in a highly varied topological setup.

V. Conclusions and Future Work

This paper presented Hypertool, an open-source, platform-agnostic framework designed to address the critical challenges of resource abstraction and dynamic node lifecycle management in heterogeneous distributed cloud environments. By implementing a dual-component architecture comprising a CLI utility and a Kubernetes-native DaemonSet, the framework successfully bridges the gap between static default orchestrator schemas and the complex, multidimensional requirements of modern workloads. The proposed declarative data model extends native node representations with rich, continuously updated attributes spanning computational capacity, energy efficiency, economic cost, operational trust, and security posture, without introducing heavy custom controllers that burden the control plane. Extensive validation across local, on-premises, and public cloud infrastructure demonstrates the framework's operational maturity, stability, and ease of integration.

Future work will focus on expanding the abstraction schema to capture deeper network topology metrics and more granular hardware accelerator telemetry (e.g., multi-GPU interconnect bandwidth). Additionally, we plan to integrate lightweight, predictive machine learning models into Hypertool's core logic to enable proactive anomaly detection and predictive node self-healing.

Acknowledgments

This work has been partially supported by the HYPER-AI project, funded by the European Commission under Grant Agreement 101135982 through the Horizon Europe research and innovation program (https://hyper-ai-project.eu/).

References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016. DOI: 10.1145/2898442.2898444. [Online]. Available: https://queue.acm.org/detail.cfm?id=2898444.

[2] S. Yang, Y. Ren, J. Zhang, J. Guan, and B. Li, "KubeHICE: Performance-aware container orchestration on heterogeneous-ISA architectures in cloud-edge platforms," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, IEEE, 2021, pp. 81–91.

[3] J. Rey-Jouanchicot, J. Á. L. Del Castillo, S. Zuckerman, and E. V. Belmega, "Energy-efficient online resource provisioning for cloud-edge platforms via multi-armed bandits," in *2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, IEEE, 2022, pp. 45–50.

[4] L. Xiao, H. Shan, J. Zhu, R. Mao, and S. Pan, "Lightweight cloud service composition and orchestration for edge intelligence," *Intelligent Decision Technologies*, vol. 19, no. 2, pp. 844–861, 2025.

[5] N. Deligiannakis et al., "DACCA: Distributed Adaptive Cloud Continuum Architecture," *Future Internet*, vol. 18, no. 74, 2 2026, ISSN: 1999-5903. DOI: 10.3390/fi18020074. [Online]. Available: https://www.mdpi.com/1999-5903/18/2/74.

[6] The Kubernetes Authors, "Daemonset | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/.

[7] The Kubernetes Authors, "Custom resources | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[8] OpenStack Foundation, "Openstack open source cloud computing software," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.openstack.org/.

[9] HashiCorp, "Nomad: Flexible workload orchestration," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.hashicorp.com/products/nomad.

[10] The OpenMetal Authors, "Manage flavors in openstack," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://openmetal.io/docs/manuals/tutorials/manage-flavors#:~:text=How%20to%20Update%20Flavor%20Metadata,select%20the%20option%20Update%20Metadata..

[11] The Kubernetes Authors, "Kubelet | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[13] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[14] Center for Internet Security, "CIS benchmarks," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.cisecurity.org/cis-benchmarks/.

[15] The MITRE Corporation, "Common vulnerabilities and exposures (CVE)," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://cve.mitre.org/.

[16] The Helm Authors, "Helm | the package manager for kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://helm.sh/.

[17] The Kubernetes Authors, "Minikube start | minikube," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://minikube.sigs.k8s.io/docs/start/.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Agentic Placement of Microservices on the Computing Continuum

Kunal Rao, Giuseppe Coviello and Srimat Chakradhar
NEC Laboratories America, Inc.
Princeton, NJ, USA
email:{kunal, giuseppe.coviello, chak}@nec-labs.com

***Abstract*—Deploying microservices across the computing continuum (edge–cloud) requires placement decisions that adapt to workload variation and heterogeneous infrastructure, yet existing solutions often rely on static policies or opaque heuristics. We present *Bellona* a system for *reliable and auditable Large Language Model (LLM)-driven workflow execution* that combines a declarative specification language with a runtime that orchestrates tool calls, conditional control flow, and structured LLM reasoning. Using *Bellona*, we implement an agentic placement workflow that automatically recommends edge or cloud execution. The workflow uses structured prompts and verifiable tool interactions to (i) parse placement and latency-report instructions, (ii) update the latency log, and (iii) select placements based on measured latency improvement thresholds. We evaluate the resulting agent on two representative microservices-based video analytics applications (human-attributes detection and face recognition) over two days of varying workload. Across 1,440 placement decisions per service, the agent achieves accuracies of 94.66%/84.94% (human-attributes detection, Day1/Day2) and 80.91%/96.53% (face recognition, Day1/Day2) with GPT-4o; with GPT-5, accuracy increases to 98.82%/99.45% (human-attributes detection) and 99.31%/99.8% (face recognition). These results demonstrate that *Bellona* can support practical, self-improving agentic control for placement of microservices on the computing continuum.**

***Keywords*-Computing continuum; Edge–cloud placement; Microservices; Agentic workflows; Large language models; Workflow orchestration; Latency-aware decision making.**

I. INTRODUCTION

The computing continuum spans resource-constrained edge nodes and elastic cloud infrastructure, enabling latency-sensitive applications while accommodating workload bursts and complex analytics. A persistent challenge in this setting is *microservice placement*: deciding, for each workload instance, whether a service should execute at the edge or in the cloud. Placement policies must contend with heterogeneous hardware, time-varying contention, and application-dependent trade-offs, and they are often deployed as static rules or hand-tuned heuristics that are brittle under changing conditions.

Recent advances in Large Language Models (LLMs) make it tempting to build "agentic" controllers that interpret operator intents, query measurements, and take actions (e.g., selecting a placement or updating state) through tool calls. In practice, however, LLM-based controllers can be difficult to operationalize: the control logic is frequently embedded in ad-hoc prompt chains, intermediate state is implicit, and the resulting behavior is hard to reproduce, debug, and audit. Existing agent frameworks (e.g., LangChain [1], Semantic Kernel [2], and DSPy [3]) simplify tool calling and composition, but they do not provide a principled substrate for specifying multi-step agent behavior with explicit state, control flow, and verifiable execution semantics.

We present *Bellona*, a system for *reliable and auditable execution of agentic workflows*. *Bellona* couples (i) a declarative, human-readable specification language for defining tasks, data dependencies, and control flow and (ii) a runtime that executes these workflows with standardized tool integration and structured LLM interactions. The specification layer supports conditional branches, iterative and parallel tasks, reusable procedures, and template-based variable substitution, while the runtime manages context, orchestrates tool calls, and enforces deterministic control flow around inherently stochastic model outputs.

To demonstrate *Bellona* on a concrete continuum-management task, we implement an agentic placement workflow that automatically recommends edge or cloud execution and incrementally learns from measurements. The workflow accepts natural-language instructions of two types: (1) requests to recommend placement for a given workload and (2) reports of observed latency for a given workload–infrastructure pair. It uses tool calls to maintain a shared Comma-Separated Values (CSV) latency log and employs structured prompts to (i) extract workload/infrastructure/latency entities, (ii) avoid duplicate records, and (iii) recommend the next placement based on the current coverage of measurements and a latency-improvement threshold once both edge and cloud observations are available.

We evaluate this agent on two representative video-analytics microservices (human-attributes detection and face recognition) under workload variation across two days and observe up to $\sim$99.8% accuracy, demonstrating that *Bellona* can support practical agentic control for placement on the computing continuum.

The main contributions of this paper are as follows:

- We present *Bellona*, a specification-first system that combines declarative workflow definitions with a runtime for reliable and auditable LLM+tool workflow execution.
- We use *Bellona* to implement an agentic placement workflow for the edge–cloud continuum that interprets natural-language instructions, maintains explicit state in a latency log, and applies a measurement-driven placement policy.
- We evaluate the resulting workflow on two video-analytics microservices under a real workload trace and show that it achieves high placement accuracy across 1,440 decisions per day.

The remainder of the paper is structured as follows: Section II discusses related work on continuum placement and agentic orchestration; Section III describes *Bellona*, including the specification language and runtime architecture; Section IV

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

presents the placement workflow implemented in *Bellona*; Section V reports experimental results and Section VI concludes.

II. Related Work

Prior work on microservice placement across the edge–cloud computing continuum spans (i) rule-based policies (e.g., fixed latency thresholds), (ii) optimization-based schedulers, and (iii) learning-based controllers that map observed load, latency, and resource availability to placement actions. In practice, these solutions are embedded in orchestration and monitoring stacks that continuously collect telemetry and trigger (re)deployment. Recent work has also begun exploring how LLMs can assist continuum-management logic, e.g., ECO-LLM [4], as well as complementary efforts on edge–cloud LLM systems and collaborative inference [5]–[7]. While effective, operationalizing these approaches often requires substantial "glue" code to connect operator intent, measurement pipelines, and application-specific objectives into repeatable decision procedures.

From a systems perspective, real-world placement is naturally a multi-step workflow: gathering measurements, updating state, evaluating a policy, and actuating changes. General-purpose workflow orchestrators, such as Apache Airflow [8], Prefect [9], Dagster [10], Temporal [11], and managed services like AWS Step Functions [12] and Azure Durable Functions [13] provide explicit control flow, retries, and scheduling for such pipelines; DagOn* [14] similarly targets DAG execution across heterogeneous resources. However, these systems largely assume deterministic task logic and do not treat LLM calls, tool-driven reasoning, or output verification as first-class components, making it difficult to build a reliable, auditable LLM-in-the-loop control loop.

LLM agent frameworks address tool use and intent interpretation by allowing models to decide which tools to call and how to compose actions. Surveys on agentic AI [15], [16] and multi-agent systems [17] describe common architectures, while systems, such as AutoGen [18], CAMEL [19], HuggingGPT [20], and CrewAI [21] demonstrate coordination across multiple LLM roles and external tools. Tool-use methods including Chain-of-Thought [22], ReAct [23], MRKL [24], Toolformer [25], and PAL [26] show how reasoning can be interleaved with actions. In many implementations, however, control flow and state transitions remain implicit in prompt chains, complicating reproducibility, debugging, and governance.

Finally, low-code automation platforms (e.g., Dify [27], n8n [28], and Windmill [29]) and LLM programming systems such as DSPy [3] and XPF [30] make it easier to compose LLM-centric pipelines. *Bellona* complements these efforts by providing a specification-first workflow substrate that (i) makes state and control flow explicit, (ii) treats LLM calls and tool calls as first-class tasks, and (iii) supports reliable, auditable execution. These properties are particularly important for operational placement workflows on the computing continuum.

III. System Design and Implementation

Bellona provides an end-to-end architecture for defining and executing *agentic workflows* in which LLM calls, tool calls, and control flow are explicit, repeatable, and auditable. Figure 1 shows the two main layers of the system: a *Specification Layer* that describes workflow structure and state dependencies, and a *Runtime Layer* that enforces deterministic control flow around inherently stochastic model outputs, orchestrates tool invocation, and records execution traces.

A. Workflow specification

Bellona workflows are written in a declarative YAML format with (i) metadata and typed inputs, (ii) labeled tasks (LLM calls, tool calls, and control-flow constructs such as conditionals/loops/parallel blocks), and (iii) reusable procedures. Jinja2-style templates (e.g., `{{instruction_type}}`) connect task outputs to subsequent inputs, making dataflow and state transitions explicit for versioning and auditability. In addition to hand-authoring YAML, *Bellona* includes a lightweight *workflow generator* that can translate a natural-language instruction into an initial workflow draft, which we then review and (if needed) minimally edit before execution.

B. Runtime execution

The runtime parses and validates the workflow, then executes tasks while maintaining a scoped *context stack* for variables across nested procedure calls. Tool invocations are mediated through a unified gateway (via MCP), which standardizes heterogeneous tools behind a common schema. For LLM tasks, the runtime expands templates, executes the model with optional tool-use loops, validates outputs when rubrics are present, and records structured traces (inputs, outputs, timing, and tool results). Parallel blocks are executed concurrently but aggregated deterministically.

Together, the specification and runtime layers turn prompt chains into an explicit program with traceable state, control flow, and tool interactions. We next describe the runtime execution engine in more detail.

C. Runtime Execution Engine

This section presents the execution semantics of *Bellona*'s runtime, which is implemented as a structured dispatcher around a context stack, a rubric-aware retry loop for LLM tasks, and a unified gateway for tool invocation. The executor coordinates templates, control flow, quality validation, caching, and observability to provide reliable and auditable execution.

1) Execution Model and Context: The runtime exposes a single entry point `Run(args, mcpCfg)` that validates the workflow, assigns task identifiers, initializes the Model Context Protocol (MCP) toolkit, and then invokes the root procedure. Each procedure call pushes a new frame on a *context stack* that stores variables, including the latest task output for dataflow chaining. Context updates use `addToContext(key, value)`, while `pushContext`/`popContext` delimit scope. Templates are

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

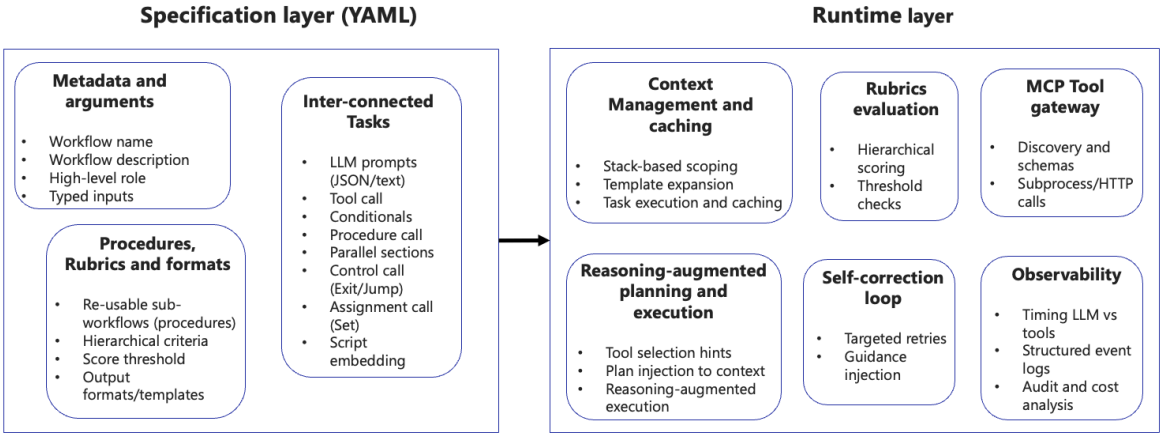


Figure 1. Overall architecture of *Bellona*. The specification layer defines YAML-based workflows (metadata, typed arguments, tasks, and reusable procedures) with optional rubrics for output validation. The runtime layer orchestrates execution using a scoped context, MCP-based tool invocation, structured LLM interactions, and trace logging for auditability.

expanded with a strict Jinja2-compatible engine before each task or tool call, which guarantees early surfacing of missing variables.

2) Task Dispatch and Control Flow: The executor visits tasks in program order and dispatches by type (LLM, tool call, conditional, procedure call, and simple state updates such as `set`). Control primitives such as `exit` and `jump` provide early termination and explicit error-recovery paths, while conditionals can be evaluated either by the LLM (for natural-language predicates) or deterministically (e.g., via a Python expression) when reproducibility is critical.

3) LLM Orchestration, Tools, and Validation: For each LLM task, the runtime expands templates, executes the model, and (when enabled) iteratively services any emitted tool calls through the MCP toolkit until the model returns a final response. When a task declares `withJSONOutput`, the runtime parses and type-checks the output before committing it to the context. If a rubric is provided, the runtime evaluates the output and may trigger a small number of guided retries using feedback derived from failed rubric criteria.

4) Caching and observability: *Bellona* supports quality-aware caching for LLM tasks: cached results are reused only when they correspond to the fully expanded prompt and satisfy any associated rubric. During execution, the runtime emits structured traces per task (prompt, tool calls, outputs, timing), which enables post-hoc auditing and debugging.

5) Determinism aids: To improve reproducibility, template expansion runs in strict mode to surface missing variables early, tool names are validated against the active MCP registry, and conditionals can be evaluated deterministically (e.g., via a Python expression) when required.

The *Bellona* runtime layer thus couples a scoped context stack, an optional rubric-driven retry loop, and an MCP-based tool gateway with explicit control primitives such as `exit` and `jump`. This combination provides reliable progress, verifiable outputs, and rich observability for LLM+tool workflows while preserving modularity through reusable procedures and typed arguments.

IV. Agentic Workflow For Placement

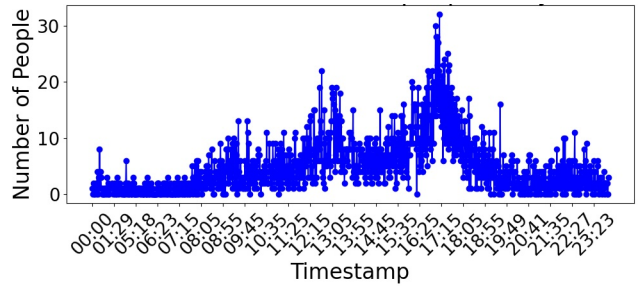
We first provide a natural-language "programming" instruction to *Bellona*'s workflow generator. The generator turns this instruction into an executable workflow that (i) interprets operator messages, (ii) maintains explicit state in a latency log, and (iii) returns an edge–cloud placement recommendation.

```
   Create a placement orchestrator that
responds with a JSON object {"placement":
"edge" or "cloud"}.    Maintain a CSV
file latency_report.csv with rows
<workload>,<infrastructure>,<latency>.
   The orchestrator receives two instruction
types:
(1) Recommend placement for workload:
<workload> and
(2) Latency on <infrastructure> for
workload: <workload> is <latency>.
   Policy: If a workload has no entries,
recommend edge. If exactly one of edge/cloud
is recorded, recommend the missing one to
collect the other measurement. If both are
recorded, recommend cloud only when it is
at least 20% faster than edge; otherwise
recommend edge.
```

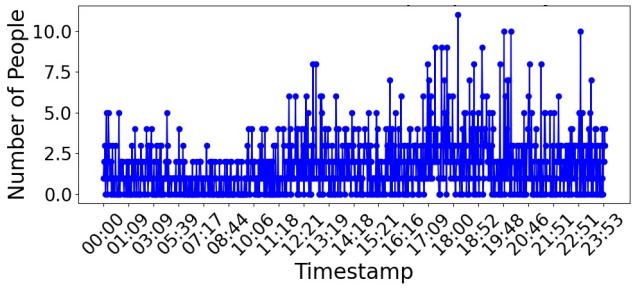
The full workflow specification used in our experiments (generated by *Bellona*'s workflow generator and then lightly edited to ensure it executes correctly) is included in Appendix A.

The workflow proceeds in four stages. It first classifies the incoming message as either a placement request or a latency report (Task `instruction_type`). For a latency report, it extracts `workload`, `infrastructure`, and `latency`, normalizes them into a CSV row, checks the existing log for duplicates, and appends the new measurement if needed. For a placement request, it reads the current log, extracts the workload identifier, and applies the policy in the prompt: cold-start on `edge`, then explore the missing infrastructure, and finally exploit `cloud` only when it exceeds the 20% improvement threshold. All state changes (file reads/appends)

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org



(a). People (Day 1)



(b). People (Day 2)

Figure 2. Variation in workload

occur through explicit tool calls, making the control loop reproducible and auditable.

V. Experimental results

A. Workload trace

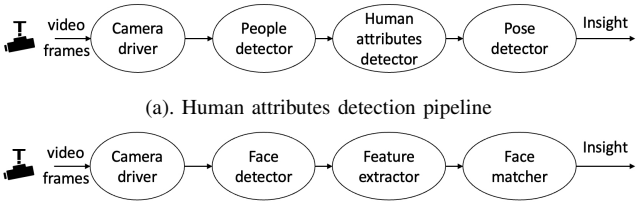
We drive the placement workflow using a real-world public trace from the City of Melbourne pedestrian counting system [31]. The trace reports the number of people observed at each sensor location over time. We select one representative location and use two 24-hour periods (one weekday and one weekend day). Figure 2 visualizes the resulting workload variation, which we treat as the per-minute input rate for both applications.

B. Applications and deployment options

Figure 3 shows the two representative microservice-based video-analytics pipelines we study: Human-Attributes Detection (HAD) and Face Recognition (FR). We consider two execution tiers: *edge* and *cloud*. Our edge tier comprises AMD Ryzen 9 5900HX, 8 cores machine, while the cloud tier uses an AWS `c4.8xlarge` VM instance. In both pipelines, the upstream microservices (`camera-driver`, `people detector`, and `face detector`) always run on the edge, while the downstream microservices (`human attributes detector`, `pose detector`, `feature extractor`, and `face matcher`) are the ones placed either on edge or on cloud.

C. Latency characterization and baseline

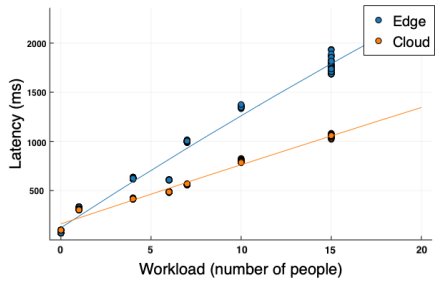
We first characterize end-to-end pipeline latency under different workloads for each placement option. In Figure 4, *Edge* denotes placement where all microservices execute on the edge machine. *Cloud* denotes placement where



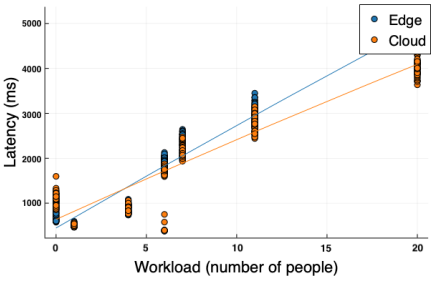
(a). Human attributes detection pipeline

(b). Face recognition pipeline

Figure 3. Representative video analytics application pipelines



(a). Human Attributes Detection



(b). Face Recognition

Figure 4. Characterization of application latency

the downstream stages execute on the cloud VM (HAD: `human attributes detector` and `pose detector`; FR: `feature extractor` and `face matcher`). We run only one pipeline/placement at a time (i.e., no concurrent pipelines).

Using these measurements, we define an *oracle baseline* that mirrors the workflow policy: for each workload value, it recommends `cloud` only if the characterized cloud latency is at least 20% lower than the characterized edge latency; otherwise it recommends `edge`. We use this baseline as the reference policy when computing workflow-driven placement decision accuracy.

D. Agentic Workflow Placement accuracy

Table I reports how often the agentic workflow matches the oracle baseline over the two days. We evaluate two LLM backends (GPT-4o and GPT-5), and we make one placement decision per minute, yielding 1,440 decisions per day (missing trace values are forward-filled with the most recent observation, and workflow is pre-warmed with latency numbers recorded for edge and cloud for different workloads).

We report (i) total decisions (TPD), (ii) incorrect `edge` recommendations when the baseline is `cloud` (TIP Edge), (iii) incorrect `cloud` recommendations when the baseline is `edge` (TIP Cloud), and the resulting accuracy. With GPT-4o,

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Table I. PLACEMENT ACCURACY OF THE AGENTIC WORKFLOW AGAINST THE ORACLE BASELINE (HAD: HUMAN ATTRIBUTES DETECTOR; FR: FACE RECOGNITION; TPD: TOTAL PLACEMENT DECISIONS; TIP: TOTAL INCORRECT PLACEMENTS).

	HAD				**FR**			
TPD	1440				1440			
	GPT-4o		GPT-5		GPT-4o		GPT-5	
	Day1	**Day2**	**Day1**	**Day2**	**Day1**	**Day2**	**Day1**	**Day2**
TIP Edge (E)	42	0	16	8	0	0	0	0
TIP Cloud (C)	35	217	1	0	275	50	10	3
TIP (E+C)	77	217	17	8	275	50	10	3
Error (%)	5.34	15.06	1.18	0.55	19.09	3.47	0.69	0.20
Accuracy (%)	**94.66**	**84.94**	**98.82**	**99.45**	**80.91**	**96.53**	**99.31**	**99.8**

the workflow achieves accuracies of 94.66%/84.94% for HAD (Day1/Day2) and 80.91%/96.53% for FR (Day1/Day2). With GPT-5, accuracy increases to 98.82%/99.45% for HAD and 99.31%/99.8% for FR.

VI. Conclusion and Future Work

In this paper, we presented *Bellona*, a specification-first system for executing agentic workflows with explicit state, conditional control flow, and verifiable tool calls, and used it to implement an LLM-driven placement workflow that (i) interprets natural-language placement and latency-report instructions, (ii) maintains a persistent CSV latency log, and (iii) recommends `edge` or `cloud` placement using a measurement-driven policy. Across two video-analytics applications and two days of workload variation, the agentic workflow achieves accuracies of 94.66%/84.94% (HAD, Day1/Day2) and 80.91%/96.53% (FR, Day1/Day2) with GPT-4o; with GPT-5, accuracy increases to 98.82%/99.45% (HAD) and 99.31%/99.8% (FR), suggesting that declarative, tool-grounded LLM workflows can provide practical continuum control while remaining reproducible and auditable. Although we demonstrate with a specific class of applications and using only 2 tiers of computing, future work can investigate the impact with different LLM models on multiple tiers of computing across a diverse set of applications.

Appendix A
Placement Orchestrator Agentic Workflow

This appendix contains the full *Bellona* workflow specification for the placement orchestrator used in our experiments.

```
title: Placement Orchestrator
description: Orchestrates placement on edge or cloud
arguments:
  instruction:
    type: string
tasks:
  - label: instruction_type
    llm:
      prompt: |
        From the instruction below, return a compact JSON
   object with the following key:
        - type: the type of the instruction, it can be
   either recommend_placement or record_latency.

        The instruction template for type
   "recommend_placement" will be like this:

        "Recommend placement for workload: <workload>"

        The instruction template for type "record_latency"
will be like this:

        "Latency on <infrastructure> for workload:
<workload> is <latency>"

        Instruction:
        {{ instruction }}
      withJSONOutput: true
      withoutTools: true
  - label: process_instruction
    conditional:
      cases:
        - condition: "{{instruction_type.type | tojson }}
== 'record_latency'"
          tasks:
            - label: extract_reported_latency
              llm:
                prompt: |
                  From the instruction below, return a
JSON object with the following keys:
                  - infrastructure: reported
infrastructure (either cloud or edge)
                  - workload: reported workload number
                  - latency: reported latency

                  The reported instruction template will
be like this:

                  "Latency on <infrastructure> for
workload: <workload> is <latency>"

                  Instruction:
                  {{ instruction }}
                withJSONOutput: true
                withoutTools: true
            - label: latency_csv
              llm:
                prompt: |
                  From the JSON below, extract and
generate a comma separated string in this format:
                  <workload>,<infrastructure>,<latency>

                  Keep the workload as an integer.

                  JSON:
                  {{ extract_reported_latency}}

                  ONLY output the comma separated string.
NOTHING ELSE.
            - label: review_prior_reported_latency
              tool_call:
                name: "readFile"
                arguments:
                  filename: "latency_report.csv"
            - label: check_if_record_exists
              llm:
                prompt: |
                  Below is a latency report in this format:
                  <workload>,<infrastructure>,<latency>

                  Reply "yes" if the specified <workload>
and <infrastructure> combination already exists in the
prior data.
                  Reply "no" if it does not exist in prior
data.

                  Each row in prior data has the same
format i.e. <workload>,<infrastructure>,<latency>

                  Reply only "yes" or "no". NOTHING ELSE.

                  latency report:
                  {{ latency_csv }}

                  Prior data:
                  {{ review_prior_reported_latency }}

            - label: record_latency
              conditional:
                cases:
                  - condition: '{{check_if_record_exists}}
== "no"'
                    tasks:
```

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

```
                    - label: append_reported_latency
                      tool_call:
                        name: "appendToFile"
                        arguments:
                          filename: "latency_report.csv"
                          text: "{{ latency_csv }}"
                    - label: respond
                      llm:
                        prompt: |
                          Respond to the user saying
that the specified latency is recorded.
              default:
                - label: respond
                  llm:
                    prompt: |
                      Respond to the user saying that
the latency for specified workload and infrastructure
combination is already recorded.

     - condition: "{{instruction_type.type | tojson }}
== 'recommend_placement'"
       tasks:
         - label: review_prior_reported_latency
           tool_call:
             name: "readFile"
             arguments:
               filename: "latency_report.csv"
         - label: extract_workload
           llm:
             prompt: |
               Extract the workload number from the
below instruction:

               Instruction:
               {{ instruction }}

               Just give the workload number. NOTHING
ELSE.

         - label: recommend_placement
           llm:
             prompt: |
               You recommend placement on edge or cloud
in JSON format. The JSON object has a single key:
               - placement: value is either "edge" or
"cloud".

               Based on the below latency data, see if
data for workload: {{extract_workload}} exists.
               DO NOT make up any data. Remember that
latency data can be empty initially.

               If it does not exist, return JSON object
with placement as "edge".

               If only one of "edge" or "cloud" data
exists, then return JSON object with placement as the
MISSING one among "edge" or "cloud".
               For example, if data for "edge" exists,
then return "cloud" and vice versa.

               If both are present, return JSON object
with placement as "cloud" ONLY IF latency improvement
over edge is greater than 20%,
               otherwise return JSON object with
placement as "edge".

               Latency data:
               {{review_prior_reported_latency}}

             withJSONOutput: true
             withoutTools: true
   default:
     - label: unknown_instruction
       llm:
         prompt: |
           Say that the instruction is not recognized.
Instruction should either report latency or request
recommendation.
```

REFERENCES

[1] LangChain, *Langchain*, https://www.langchain.com/, Last accessed Feb. 25, 2026.

[2] Microsoft, *Semantic Kernel documentation*, [https://learn.microsoft.com/en-us/semantic-kernel/], Last accessed Feb. 25, 2026, 2025.

[3] O. Khattab *et al.*, "DSPy: Compiling declarative language model calls into self-improving pipelines", 2024.

[4] K. Rao *et al.*, "Eco-llm: Llm-based edge cloud optimization", in *Proceedings of the 2024 Workshop on AI For Systems*, ser. AI4Sys '24, Pisa, Italy: Association for Computing Machinery, 2024, pp. 7–12, ISBN: 9798400706523. DOI: 10.1145/3660605.3660941.

[5] Z. Zhang *et al.*, *Creating edge ai from cloud-based large language models*, arXiv preprint arXiv:2410.18080, 2024.

[6] J. Chen *et al.*, *Edgeshard: Efficient llm inference on edge devices via sharded kv cache*, arXiv preprint arXiv:2405.14371, 2024.

[7] J. Xu *et al.*, *Ce-collm: Computation-efficient collaborative large language model inference for edge-cloud environments*, arXiv preprint arXiv:2411.02829, 2024.

[8] *Apache airflow*, https://airflow.apache.org/, Last accessed Feb. 25, 2026.

[9] *Prefect documentation*, https://docs.prefect.io/latest/, Last accessed Feb. 25, 2026.

[10] *Dagster documentation*, https://docs.dagster.io/, Last accessed Feb. 25, 2026.

[11] *Temporal*, https://temporal.io/, Last accessed Feb. 25, 2026.

[12] *Aws step functions developer guide*, Last accessed Feb. 25, 2026, Amazon Web Services.

[13] *Azure durable functions overview*, Last accessed Feb. 25, 2026, Microsoft.

[14] R. Montella, D. Di Luccio, and S. Kosta, "Dagon*: Executing direct acyclic graphs as parallel jobs on anything", in *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, IEEE, 2018, pp. 64–73.

[15] D. B. Acharya, K. Kuppan, and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey", *IEEE Access*, vol. 13, pp. 18 912–18 936, 2025. DOI: 10.1109/ACCESS.2025.3532853.

[16] A. K. Pati, "Agentic AI: A Comprehensive Survey of Technologies, Applications, and Societal Implications", *IEEE Access*, vol. 13, pp. 151 824–151 837, 2025. DOI: 10.1109/ACCESS.2025.3585609.

[17] S. Han, Q. Zhang, Y. Yao, W. Jin, and Z. Xu, *Llm multi-agent systems: Challenges and open problems*, 2025. arXiv: 2402.03578 [cs.MA].

[18] *Autogen: Enabling next-gen llm applications via multi-agent conversation framework*, 2023. arXiv: 2308.08155.

[19] *Camel: Communicative agents for "mind" exploration*, 2023. arXiv: 2303.17760.

[20] *HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace*, 2023. arXiv: 2303.17580.

[21] *Crewai documentation*, https://docs.crewai.com/, Last accessed Feb. 25, 2026.

[22] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models", *Neural Inf Process Syst*, vol. abs/2201.11903, Jan. 2022.

[23] *React: Synergizing reasoning and acting in language models*, 2022. arXiv: 2210.03629.

[24] *Mrkl systems*, 2022. arXiv: 2205.00445.

[25] *Toolformer: Language models can teach themselves to use tools*, 2023. arXiv: 2302.04761.

[26] *Program-aided language models*, 2022. arXiv: 2211.10435.

[27] *Dify*, https://dify.ai/, Last accessed Feb. 25, 2026.

[28] *N8n*, https://n8n.io/, Last accessed Feb. 25, 2026.

[29] *Windmill*, https://www.windmill.dev/, Last accessed Feb. 25, 2026.

[30] K. Rao, G. Coviello, G. Mellone, C. G. De Vita, and S. Chakradhar, "XPF: Agentic AI System for Business Workflow Automation", in *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '25, University of Notre Dame Conference Facilities, Notre Dame, IN, USA: Association for Computing Machinery, 2025, pp. 1–6, ISBN: 9798400718694. DOI: 10.1145/3731545.3743644.

[31] City of Melbourne, *Pedestrian Counting System — Past Hour (counts per minute)*, https://melbournetestbed.opendatasoft.com/explore/dataset/pedestrian-counting-system-past-hour-counts-per-minute/information/?sort=-location_id, Last accessed Feb. 25, 2026, 2022.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

LLM Fine-Tuning with aiDAPTIV+: A Viable Training Strategy on Resource-Constrained Compute Platforms

Abhishek Patel, Krishan Gopal Gupta, Shashank Sharma, Sowmya Shree, Sanjay Wandhekar
Centre for Development of Advanced Computing (C-DAC), Pune, India
Email: {abhishekp, krishang, shashank.sharma, ssowmya, sanjayw}@cdac.in

***Abstract*—The paper addresses the challenge of fine-tuning Large Language Models (LLMs) on resource-constrained hardware. The need for effective fine-tuning solutions has grown dramatically due to the quick development of open-source LLMs like the Large Language Model Meta AI (LLaMA). For researchers and practitioners with limited hardware, training these models is costly and difficult due to the significant computational resources needed. In order to enable and improve LLM fine-tuning performance, this study investigates an alternative method that makes use of MiPhi aiDAPTIV+ AI Solid-State Drives (SSDs). The impact of these SSDs is compared to traditional dense Graphics Processing Unit (GPU)-accelerated training setups that exclusively rely on the memory and compute power of high-end GPUs. The primary objective is to investigate whether AI SSDs can help older High Performance Computing (HPC) clusters and workstations overcome memory and processing constraints. The methodology is equally applicable to public cloud environments (e.g., AWS, Azure, GCP), where GPU instances are often constrained by cost, memory and processing constraints, making it easier to run contemporary LLM workloads even with constrained GPU resources. Llama 2 and Llama 3 models were fine-tuned with a precision of Brain floating point 16 (BF16) using extensive experiments on the Center for Development of Advanced Computing's (CDAC) PARAM Rudra HPC platform and PARAM Siddhi AI platform, both of which had NVIDIA A100 40GB GPUs. Using the DeepSpeed framework, the effects of MiPhi aiDAPTIV+ AI SSDs and their middleware are evaluated against a dense GPU system with an emphasis on increases in data transfer speeds, model checkpointing and overall system utilization. The study shows that the feasible configuration space for LLM fine-tuning on modest hardware is greatly expanded by AI SSD-accelerated training, allowing models that would otherwise fail due to memory exhaustion to run successfully on minimal GPU configurations.**

Keywords-LLM; fine-tuning; MiPhi aiDAPTIV+; HPC; Token Throughput Optimization.

I. INTRODUCTION

Natural Language Processing (NLP) has undergone a significant transformation with the emergence of LLMs [**2**], enabling advanced applications across domains, such as software engineering, healthcare, finance, customer service, and scientific research [**1**]. Recent open-source models, including Meta's LLaMA series and the DeepSeek family, have demonstrated strong capabilities in few-shot learning, reasoning, and coherent text generation. These capabilities are largely driven by large-scale pre-training on diverse corpora followed by fine-tuning on domain-specific datasets. As organizations increasingly integrate LLMs into critical workflows, the demand for cost-effective and resource-efficient training methodologies has grown substantially.

Despite these advancements, modern LLM training and fine-tuning introduce several system-level challenges [**3**]:

- **High GPU Memory Requirements:** Even relatively smaller LLM variants with billions of parameters require GPUs with large High Bandwidth Memory (HBM), making them incompatible with older-generation hardware.
- **Infrastructure Bottlenecks:** The cost and complexity of deploying next-generation GPU infrastructure pose significant barriers for research institutions, government laboratories, and startups.
- **Limitations of Legacy HPC Systems:** Traditional High Performance Computing (HPC) clusters, primarily optimized for simulation workloads, often struggle with the memory bandwidth and I/O patterns required for LLM training.
- **Scaling Challenges:** Increasing batch sizes and model complexity impose additional pressure on system architecture, requiring efficient data access, parameter offloading, and high-bandwidth memory management for optimizer states.
- **Framework Constraints:** While frameworks such as DeepSpeed, Hugging Face Accelerate, and parameter-efficient techniques like Low-Rank Adaptation (LoRA) improve computational efficiency, their performance remains constrained by storage transfer rates and I/O bottlenecks.
- **Need for Alternative Architectures:** These limitations highlight the necessity of exploring alternative storage and memory hierarchies, such as SSD-accelerated workflows and burst buffer integration, to mitigate system bottlenecks.

To address these challenges, this paper investigates MiPhi AI-accelerated SSDs as an alternative to conventional GPU-centric training architectures. Unlike standard NVMe SSDs, MiPhi AI SSDs, integrated with the aiDAPTIV+ middleware, employ a co-designed hardware-software approach optimized for transformer-based workloads. The system dynamically partitions model states and offloads less frequently accessed data from GPU memory to high-speed SSD storage, effectively expanding usable memory capacity. This enables efficient data prefetching, real-time checkpointing, and high-throughput training without requiring modifications to existing training pipelines.

The performance of MiPhi aiDAPTIV+ SSDs [**4**] is evaluated during fine-tuning of LLaMA 2 and LLaMA 3 models

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

using BF16 precision. The study focuses on analyzing improvements in I/O throughput and training efficiency across both single-GPU and multi-GPU environments. The key contributions of this work are summarized as follows:

- **Empirical Evaluation:** A comprehensive benchmark assessing the impact of AI-accelerated SSDs on data loading, checkpointing, and overall system throughput during LLM fine-tuning.
- **Single-GPU Optimization:** An analysis of performance gains achievable for researchers operating under limited hardware constraints.
- **Cost-Performance Analysis:** A comparative study between GPU-only and AI SSD-accelerated setups to guide cost-efficient infrastructure decisions for LLM training.
- **Configuration Space Characterization:** A systematic mapping of feasible LLM fine-tuning configurations (model size, batch size, sequence length, GPU count) enabled by AI SSD offloading on production HPC nodes, providing practitioners with a reproducible methodology applicable to both HPC and public cloud environments.

The remainder of this paper is organized as follows. Section 2 reviews recent advancements in LLM training methodologies, including parameter-efficient fine-tuning, offloading strategies, distributed training, and quantization techniques. Section 3 presents the MiPhi aiDAPTIV+ architecture, detailing its hardware design and middleware components. Section 4 describes the experimental setup, including system configurations, datasets, and workload parameters. Section 5 outlines the execution methodology. Sections 6 and 7 present performance results and throughput benchmarks on PARAM Siddhi [**5**] and PARAM Rudra systems [**6**]. Section 8 discusses key observations and technical insights, and Section 9 concludes the paper with directions for future work.

II. Training Requirements of LLMs: Current State of the Art

Training and fine-tuning of LLMs have become essential for adapting general-purpose foundation models to domain-specific applications. While models such as LLaMA, GPT, and DeepSeek are pre-trained on large and diverse datasets, further fine-tuning is required to incorporate specialized knowledge, enforce organizational constraints, and align with user-specific requirements. However, this process introduces significant computational and memory challenges.

To address these limitations, several optimization techniques have been developed:

- **Parameter-Efficient Fine-Tuning (PEFT):** PEFT techniques, including Low-Rank Adaptation (LoRA), Quantized LoRA (QLoRA), and Adapter-based methods, enable fine-tuning by introducing a small number of trainable parameters into pre-trained transformer layers [**7**][**8**][**9**]. By keeping the majority of model weights frozen, these methods significantly reduce memory and computational requirements, allowing LLM fine-tuning on commodity GPUs with limited memory capacity.
- **Offloading and CPU-GPU Hybridization:** To further reduce GPU memory pressure, techniques, such as ZeRO-Offload in DeepSpeed and hybrid execution pipelines in Hugging Face Accelerate move optimizer states and intermediate activations to CPU memory or NVMe storage [**10**]. Frameworks like Colossal-AI [**14**] extend this concept by enabling hierarchical memory management across CPU and GPU tiers.
- **Sharded and Distributed Training:** Large-scale training is facilitated through distributed strategies such as Fully Sharded Data Parallel (FSDP) and Megatron-DeepSpeed, which partition model parameters, gradients, and optimizer states across multiple devices [**11**]. These approaches reduce per-device memory requirements and enable multi-node scalability, but introduce communication overhead and increased system complexity.
- **Mixed Precision and Quantization:** The use of reduced precision formats such as BF16 and INT8 significantly lowers memory consumption and improves computational throughput. Techniques such as Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ) enable efficient deployment of large models with minimal accuracy degradation [**12**][**13**]. Notably, QLoRA employs 4-bit quantization to achieve substantial memory savings, enabling fine-tuning of models with billions of parameters on limited hardware resources.

Despite these advancements, training and fine-tuning of LLMs continue to face substantial computational challenges. These include managing parameter updates, maintaining optimizer states, and performing checkpointing within constrained GPU memory environments.

For instance, full-precision fine-tuning of a 7B parameter model such as LLaMA 2 using the Adam optimizer can require over 100 GB of GPU memory due to the storage of model parameters, gradients, and optimizer states, which together can consume approximately 16 bytes per parameter [**16**]. Additionally, large-scale training pipelines demand high-throughput storage systems, often requiring terabytes of capacity for data preprocessing, tokenization, checkpointing, and evaluation. Consequently, memory bandwidth and I/O throughput remain critical bottlenecks in efficient LLM training.

III. MiPhi aiDAPTIV+ Architecture

MiPhi's aiDAPTIV+ is a combined hardware and software system created to address memory issues during the LLM's training. The sections that follow explain both the hardware architecture and the middleware stack.

A. aiDAPTIV+ Hardware Architecture and Integration

The main hardware element features an AI100E SSD, offered in both U.2 and M.2 formats, that functions as an aiDAPTIVCache to expand GPU memory. It supports up to 320 GB for personal computers and 8 TB for workstations or servers and provides up to 100 Drive Writes Per Day (DWPD) using Single Level Cell (SLC) NAND along with advanced NAND correction algorithms, allowing for the operation of

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

transformer-based models such as LLaMA 3 70B and Falcon 180B on a single server.

B. aiDAPTIV+ Middleware and Software Stack

The aiDAPTIV+ software architecture is intended to allow seamless integration into current AI workflows, especially those developed using the PyTorch framework. The middleware layer aiDAPTIVLink functions as a virtual memory controller, enabling smooth interaction between the AI application and the underlying aiDAPTIVCache hardware. The aiDAPTIV+ middleware library extends available GPU memory through dynamic memory virtualization using SSDs, enabling the offloading of tensors and model parameters. This enables the training and inference of large transformer-based models like LLaMA and Mistral within current PyTorch-based workflows, without the need to alter the training process.

The architecture is structured to support high-speed, low-latency I/O operations and precise management of data placement. The platform can be accessed through both a Command-Line Interface (CLI) and a graphical user interface (aiDAPTIVPro Suite) and it supports workflows such as data ingestion and fine-tuning.

IV. EXPERIMENTAL SETUP

This section describes the system configuration, datasets, and experimental parameters used to evaluate MiPhi aiDAPTIV+ performance.

A. Compute Infrastructure Overview

To evaluate the impact of MiPhi AI SSD acceleration on LLM fine-tuning, experiments were conducted on single node of two high-performance computing (HPC) platforms: **PARAM Siddhi AI** and **PARAM Rudra**. Their summarized node-level specifications are shown in Table I.

These platforms played complementary roles: **PARAM Siddhi AI (TestBed 1)** offered a dense multi-GPU environment, while **PARAM Rudra (TestBed 2)** represented a typical 2-GPU server setup.

1) System Software Stack Overview: Table II summarizes the software environment used on the PARAM Rudra and PARAM Siddhi AI clusters. These software configurations are crucial for ensuring compatibility and performance during large-scale training using MiPhi AI SSD middleware.

TABLE I. SYSTEM SPECIFICATIONS OF PARAM SIDDHI AI (TESTBED 1) AND PARAM RUDRA (TESTBED 2).

Specification	**PARAM Siddhi AI (TestBed 1)**	**PARAM Rudra (TestBed 2)**
CPU	Dual AMD EPYC 7742	Dual Intel Xeon Gold 6240R
Cores/Threads per Node	128 cores, 256 threads	48 physical cores
Clock Speed	2.25 GHz	2.40 GHz
L3 Cache	256 MB	36 MB
System Memory (RAM)	1 TB per node	192 GB per node
GPU	8× NVIDIA A100 (SXM4, 40 GB each)	2× NVIDIA A100 (PCIe Gen3, 40 GB each)
GPU Memory per Node	320 GB	80 GB
MiPhi AI SSD	4 TB	2 TB
Networking	Mellanox ConnectX-6 VPI (InfiniBand HDR), 1.6 Tbps	InfiniBand HDR fabric
Shared Storage	10.5 PiB Lustre parallel file system	1 PiB Lustre parallel file system

TABLE II. SOFTWARE STACK ON PARAM RUDRA AND PARAM SIDDHI AI.

Component	**PARAM Rudra**	**PARAM Siddhi AI**
OS	Ubuntu 22.04.5 LTS	Ubuntu 20.04.2 LTS
Kernel Version	5.15.0-142-generic	5.4.0-80-generic
Architecture	x86_64	x86_64
NVIDIA Driver	550.163.01	450.142.00

B. Workload Configurations

The study focused on fine-tuning various Llama 2 model sizes 7B, 13B, and 70B across different system configurations. The dataset Cohere/miracl-zh-queries-22-12 [**17**] was used for this purpose. The dataset is designed for multilingual retrieval tasks and features natural language queries that closely mimic real-world usage, especially in information retrieval and question-answering domains.

The dataset is loaded using a 100% training split (no validation or test subsets) and models are trained with varying batch sizes and sequence lengths to evaluate system throughput and fine-tuning efficiency. The training parameters used include:

- Batch Sizes: {8, 16, 32, 64, 128}
- Sequence Lengths: {128, 256, 512, 1024, 2048}
- Model Sizes: {7B, 13B, 70B}
- Nodes: {1, 2, 4, 8} across PARAM Rudra and PARAM Siddhi
- Precision: BF16

The choice of batch sizes and sequence lengths in powers of two minimizes memory fragmentation and improves GPU utilization, which directly impacts training throughput and system stability. Similarly, exploring all permutations of model size, batch size and sequence length allows a thorough evaluation of training scalability and system behavior under diverse conditions. This comprehensive matrix helps identify hardware bottlenecks, such as memory exhaustion or

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

data transfer latencies, particularly relevant when comparing traditional storage versus AI SSD acceleration.

V. Execution Workflow

The experimental process is intended to assess how the performance of LLM fine-tuning changes when using single-node, multi-GPU configurations with MiPhi aiDAPTIV+ technology compared to a standard DeepSpeed setup. The process takes place in the following manner:

- **Data Preparation and Staging:** Pre-tokenized datasets are loaded into the MiPhi AI SSD at the start, minimizing host-to-device I/O latency during subsequent epochs.
- **Model Initialization:** Llama variants (7B, 13B, 70B) are instantiated in GPU memory using BF16 precision, ensuring sufficient headroom for batch-based workloads.
- **Activation Offloading:** The aiDAPTIV+ middleware offloads and prefetches activations to and from the SSD concurrently with ongoing GPU computation, allowing I/O transfers to overlap with arithmetic operations.
- **Checkpointing and Streaming:** Checkpoints are periodically written to the SSD with minimal I/O stalling, enabled by high-throughput, low-latency access in conjunction with intelligent storage scheduling.
- **Performance Monitoring:** Throughout training, key performance metrics such as tokens per second (throughput), micro-batch latency, and loss convergence are logged to quantitatively evaluate the efficacy of the aiDAPTIV+ stack.

By offloading memory pressure from GPU DRAM to SSD and overlapping data movement with computation, this workflow maintains high utilization even when training large-context LLMs (7B-70B) a challenge often highlighted in recent SSD offloading research [**15**]. Key training parameters such as batch size and sequence length are varied, keeping precision as BF16, to test the boundaries of model size that can be trained given the number of GPUs.

VI. Results and Observations

This section presents the experimental results from both PARAM Siddhi and PARAM Rudra platforms, evaluating the impact of MiPhi aiDAPTIV+ SSD on GPU memory efficiency, model scalability, and throughput.

A. PARAM Siddhi: With & Without MiPhi aiDAPTIV+ SSD

The primary objective was to evaluate the impact of MiPhi AI SSD on GPU memory efficiency, model scalability, and throughput improvement across varying model sizes and configurations.

1) PARAM Siddhi: Without MiPhi aiDAPTIV+ SSD: A set of experiments was carried out on the PARAM Siddhi HPC system a high-density GPU system to assess the training potential of LLaMA 2 models of different sizes (7B, 13B, and 70B), both with and without MiPhi aiDAPTIV+ SSD optimization. In baseline configuration, without MiPhi, we employed BF16 precision and DeepSpeed ZeRO-3 optimization to enable memory-efficient fine-tuning. These experiments allowed us to identify the minimum GPU requirements and limitations when MiPhi acceleration was not used, which served as a comparative baseline MiPhi's impact.The configurations were:

TABLE III. Minimum GPU Requirements Without MiPhi aiDAPTIV+ SSD for LLaMA 2 Models.

Model	Batch Size	Seq Length	Minimum GPUs Required W/o MiPhi AI SSd
LLaMA 2 – 7B	4	2048	4 GPUs
LLaMA 2 – 13B	4	512	7 GPUs
LLaMA 2 – 70B	1	–	> Unable to fit even with 8 GPUs

Table III summarizes the minimum GPU requirements, and Figure 1 visualizes the corresponding training efficiency across GPU counts.

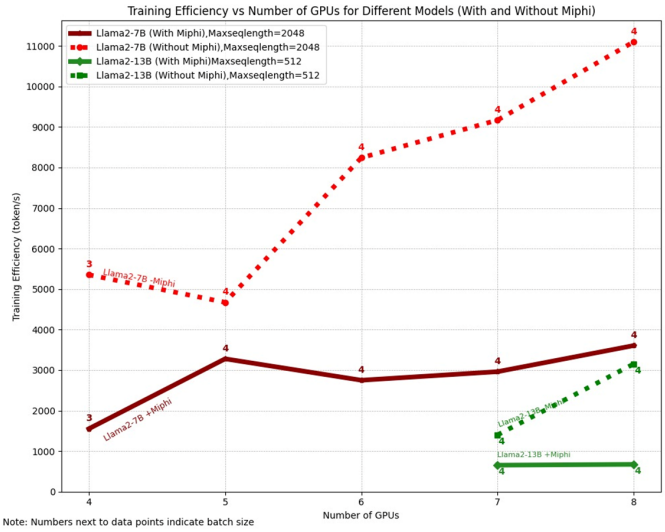


Figure 1. Training efficiency vs. number of GPUs for different Llama-2 models with and without MiPhi aiDAPTIV+ SSD. Numbers indicate batch sizes.

Figure 1 illustrates the training efficiency across varying GPU counts for LLaMA 2 models on PARAM Siddhi without MiPhi aiDAPTIV+ SSD. Key findings:

- **LLaMA 2 – 7B:** Scaled well from 4 to 8 GPUs, reaching over 11, 000 tokens/sec.
- **LLaMA 2 – 13B:** Required at least 7 GPUs and showed diminishing returns.
- **LLaMA 2 – 70B:** Failed to fit on 8 GPUs, confirming its infeasibility without memory offloading.
- **LLaMA 2 – 7B:** Achieved stable scaling from 4 to 8 GPUs at sequence length 2048, increasing throughput

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

from ∼5000 to over 11, 000 tokens/sec. Minimum 4 GPUs were required to fit the model.
- **LLaMA 2 – 13B:** Required at least 7–8 GPUs even at sequence length 512. Throughput remained under 3000 tokens/sec, showing diminishing returns with increasing GPU count.
- **LLaMA 2 – 70B:** Failed to fit on 8 GPUs even with minimum batch size and sequence length, indicating infeasibility without memory offloading.

These observations confirm that while smaller models scale efficiently without MiPhi aiDAPTIV+ SSD, larger models (13B and above) quickly encounter memory bottlenecks emphasizing the necessity of AI SSD-assisted training infrastructure for modern LLM workloads.

2) PARAM Siddhi: Training Efficiency and Scaling Performance With MiPhi aiDAPTIV+ SSD: To assess the scalability and efficiency of MiPhi-accelerated LLM training, a controlled set of experiments was conducted by fixing the sequence length to 2048 tokens and varying the batch size and number of GPUs. These experiments were conducted using LLaMA 2 models (7B and 13B) on PARAM Siddhi with MiPhi aiDAPTIV+ integration. Training efficiency is measured in tokens per second and does not indicate model accuracy or loss. The results highlight:

- Higher batch size improves throughput but not necessarily convergence or optimal training.
- Model performance remains reliant on proper hyperparameter tuning.

Figure 2 presents the training throughput for LLaMA 2 models using MiPhi aiDAPTIV+ acceleration across different GPU counts.

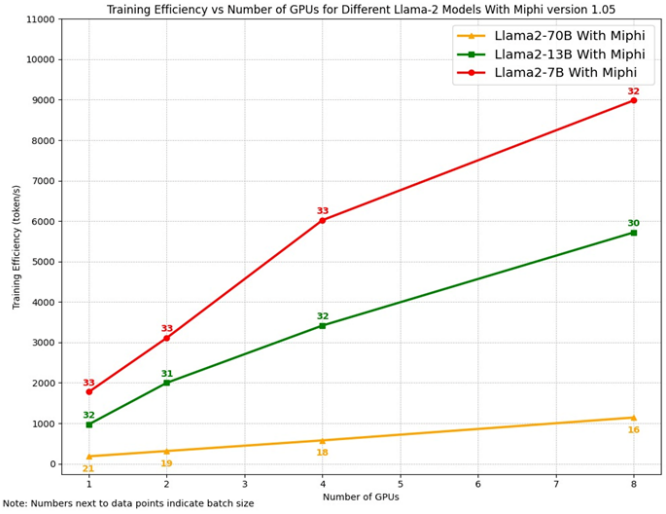


Figure 2. Training efficiency vs. number of GPUs using MiPhi aiDAPTIV+ v1.05 with maximum batch size. Linear scaling observed up to 8 GPUs.

Key insights:

- **LLaMA 2 – 7B:** Achieved linear scaling up to 8 GPUs with training efficiency exceeding 9000 tokens/sec. Batch size remained consistent (∼32–33), indicating effective memory offloading via MiPhi aiDAPTIV+.
- **LLaMA 2 – 13B:** Demonstrated steady scaling from 1 to 8 GPUs, reaching 5000+ tokens/sec. MiPhi aiDAPTIV+ enabled consistent batch sizes (∼30–32), which are infeasible on traditional setups without memory augmentation.
- **LLaMA 2 – 70B:** Successfully trained on as few as 1–2 GPUs with MiPhi aiDAPTIV+ (batch sizes 19-21), scaling up to 1000+ tokens/sec on 8 GPUs. Without MiPhi aiDAPTIV+ SSD, this model could not be trained on a single node due to memory limitations.

Overall, MiPhi aiDAPTIV+ SSD improved scalability and training feasibility across model sizes by enabling larger batch sizes, stable throughput, and efficient memory utilization, even for massive models like LLaMA 2-70B.

B. PARAM Rudra: With MiPhi aiDAPTIV+ SSD

The evaluation was further extended to the PARAM Rudra platform a smaller GPU system which features a single node equipped with 2×NVIDIA A100 GPUs. In this setup, a single 2 TB MiPhi aiDAPTIV+ AI100E SSD (U.2 form factor) was integrated and experiments were conducted to assess the feasibility and efficiency of LLM fine-tuning using both 1-GPU and 2-GPU configurations.

Training was performed using the MiPhi aiDAPTIV+ stack with middleware versions 1.5 and 2.0. Version 1.5 was tested for scaling behavior across varying batch sizes for LLaMA 2 models, while version 2.0, which provides support for newer foundation models, enabled successful fine-tuning of the **LLaMA 3.1–8B** model.

1) Training Requirements and Observations: **Hardware Configuration:** All experiments were conducted on the PARAM Rudra Server Board using the aiDAPTIV+ v2.0 Kit. Training was performed using Llama 2 and Llama 3.1 variants with the following configuration:

- **Dataset:** `Cohere/miracl-zh-queries-22-12`
- **Batch Sizes Tested:** 8, 16, 24, 32
- **Maximum Sequence Lengths:** 128, 256, 512, 1024, 2048

Figure 3 and Figure 4 show token throughput for LLaMA 2-7B and LLaMA 3.1-8B respectively on a single GPU on PARAM Rudra, across varying batch sizes and sequence lengths.

The performance graphs demonstrate token throughput for LLaMA 2-7B and LLaMA 3.1-8B models using a single NVIDIA A100 GPU on the PARAM Rudra platform. For LLaMA 2-7B, throughput scales consistently with both increasing sequence length and batch size, achieving over 1100 tokens/sec at the maximum configuration (batch size 32, sequence length 2048). This indicates effective utilization of GPU resources for smaller models.

In contrast, LLaMA 3.1-8B shows more constrained scaling behavior. The model faces memory limitations beyond sequence lengths of 512 for batch sizes larger than 16, preventing data collection for higher configurations. Nonetheless, for supported configurations, performance trends upward, reaching close to 500 tokens/sec for batch size 8 and sequence length 2048.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

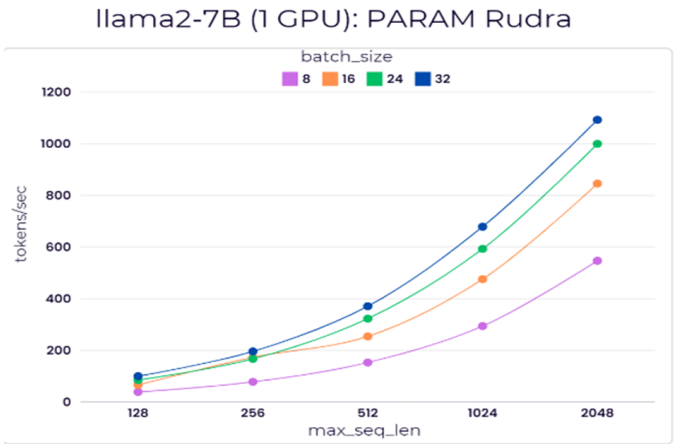


Figure 3. Llama 2 (7B), 1 GPU on PARAM Rudra: Scaling efficiency increased with increase in batch size and sequence length

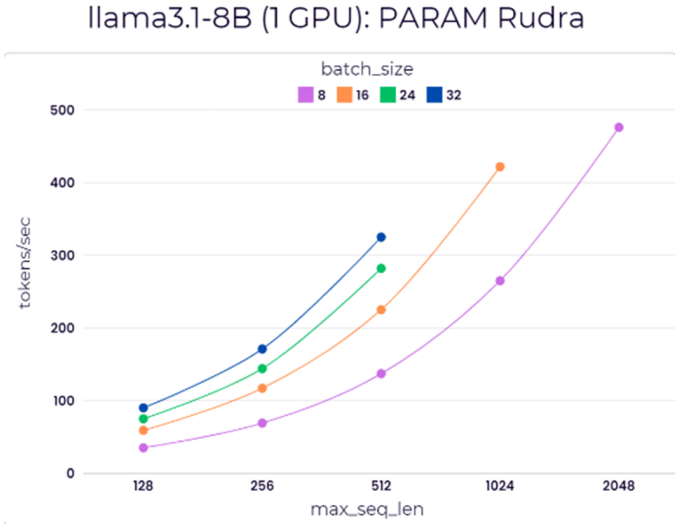


Figure 4. Llama 3.1 (8B), 1 GPUs with MiPhi aiDAPTIV+ on PARAM Rudra. Experiments failed on large batch size and sequence length combinations.

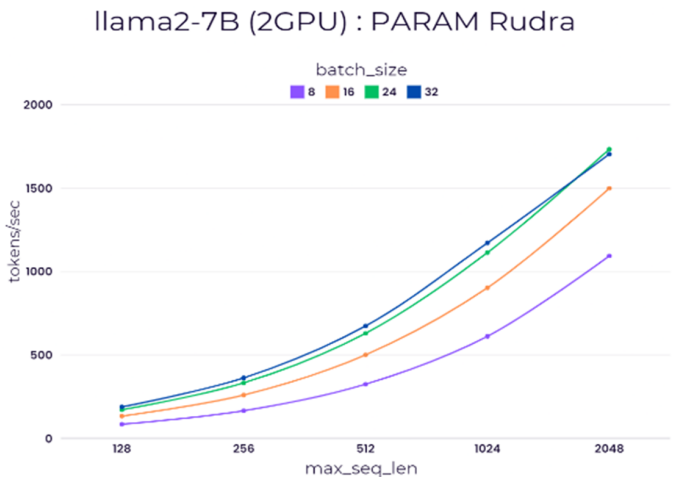


Figure 5. Llama 2 (7B), 2 GPUs with MiPhi aiDAPTIV+ on PARAM Rudra.

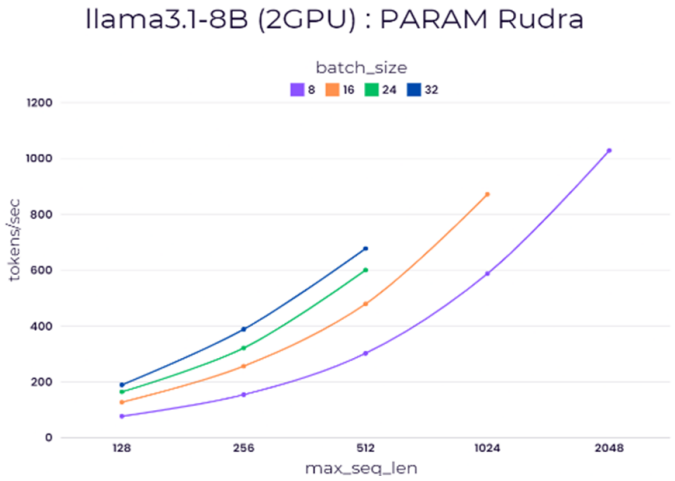


Figure 6. Llama 3.1 (8B), 2 GPUs with MiPhi aiDAPTIV+ on PARAM Rudra. Experiments failed on large batch size and sequence length combinations.

To determine the minimum GPU requirements for training the LLaMA 3.1-8B model with a larger maximum sequence length, experiments were conducted using a 2-GPU configuration for further evaluation. As shown in Figure 5 and Figure 6, experiments conducted using two GPUs revealed similar behavior to the single-GPU configuration, with no significant changes observed in performance when increasing the number of GPUs.

While scaling in terms of **training efficiency** (tokens/sec) was observed as the number of GPUs increased, several configurations failed due to **CUDA Out of Memory (OOM)** errors. **Failed Experiment Configurations for LLaMA 3.1-8B(2GPUs):**

- **Max Sequence Length = 1024:** Batch sizes of 24 and 32
- **Max Sequence Length = 2048:** Batch sizes of 16, 24, and 32

These observations suggest that, even with multiple GPUs, memory constraints remain a key bottleneck. This highlights the need for further investigation into the extent to which large models can be accommodated using **NVMe-based memory extension** solutions such as MiPhi's AI SSD.

C. Comparative Analysis of MiPhi aiDAPTIV+ v2.0 on PARAM Siddhi and PARAM Rudra

To evaluate the performance of **MiPhi aiDAPTIV+ v2.0**, benchmarking was carried out on two supercomputing infrastructures: **PARAM Siddhi** and **PARAM Rudra**. The experiments used a maximum sequence length of 2048 tokens across various GPU and batch size configurations.

The key hardware specifications of the two systems are summarized in Table IV. This highlights differences in GPU interconnect, NVMe configuration, aiDAPTIV kit capacity, and system RAM, which directly influence the observed throughput variations.

Following the hardware overview, the throughput in tokens per second (`tokens/sec`) was recorded for varying GPU

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE IV. HARDWARE DIFFERENCES: PARAM SIDDHI VS. RUDRA.

Component	Siddhi	Rudra
GPU Interface	SXM4 (600 GB/s)	PCIe Gen3 (8 GB/s)
NVMe Config	RAID 0	Non-RAID
aiDAPTIV kits	2 × 2 TB	1 × 2 TB
System RAM	1 TB	192 GB

and batch size configurations, as shown in Table V and visualized in Figure 7.

D. Results: Training Throughput on PARAM Rudra

Benchmarking experiments were performed on the LLaMA-2 7B and LLaMA-3.1 8B models using PARAM Rudra with both single-GPU and dual-GPU configurations. The experiments involved testing different combinations of batch sizes and sequence lengths to evaluate the resulting throughput and to analyze the impact of increasing maximum sequence length at various batch sizes. Additionally, these tests allowed identification of the practical limitations of fine-tuning larger models, specifically determining the batch size and sequence length at which training fails due to CUDA OOM errors. A summary of these benchmarking results is presented in Table VI.

TABLE V. TOKENS/SEC FOR DIFFERENT GPU AND BATCH SIZE CONFIGURATIONS ON SIDDHI AND RUDRA (MAX_SEQ_LEN = 2048)

GPUs (Batch Size)	Siddhi (tokens/sec)	Rudra (tokens/sec)
1 (18)	1803	914
1 (33)	1776	1180
2 (18)	3806	1451
2 (33)	3235	1739

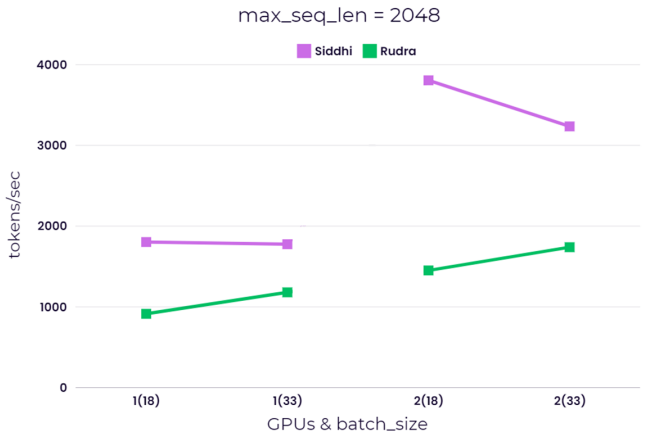


Figure 7. Performance comparison of MiPhi aiDAPTIV+ v2.0 on PARAM Siddhi and Rudra with varying GPU and batch size configurations (max_seq_len = 2048).

Using **MiPhi aiDAPTIV+ v2.0**, the fine-tuning performance of the LLaMA-3.1 70B model on **PARAM Rudra**, was evaluated, aiming to determine the maximum batch size achievable and how throughput scales with increasing batch size. In Figure 8, the token efficiency (tokens/sec) was empirically evaluated for fine-tuning the LLaMA-3.1 70B model on a single PARAM Rudra node using one and two NVIDIA A100 (40GB) GPUs, paired with a 2 TB MiPhi AI DAPTIV SSD. The evaluation spanned batch sizes from 1 to 80, with a fixed sequence length of 256 tokens.

- With a single GPU, the peak token efficiency achieved was **52.3 tokens/sec** at batch size 80.
- With both GPUs utilized, efficiency rose to a maximum of **86.9 tokens/sec**, marking a **65.9% improvement** over the single-GPU case.
- At 52.3 tokens/sec, training on a dataset of 10 billion tokens would require approximately **2.22 days** on a single GPU (assuming uninterrupted operation).
- With 2 GPUs and peak efficiency of 86.9 tokens/sec, the same workload would complete in **1.33 days**.
- This aligns with token-based training time estimates reported by Meta for Llama training runs, which cite token throughput as a core determinant of training duration and hardware planning.
- In academic labs or small HPC centers, limited compute resources slow down model training and hinder rapid experimentation.
- Multi-GPU scaling boosts throughput, but the costs of hardware, energy, and maintenance often outweigh the gains unless paired with efficiency methods such as mixed-precision training, parameter-efficient tuning (e.g., LoRA), or quantization.
- Overall, these experiments contribute to the discourse on **democratizing LLM training and inference**, underscoring the relevance of I/O-aware system design and cost-effective hardware strategies for practical deployment across constrained computing environments.

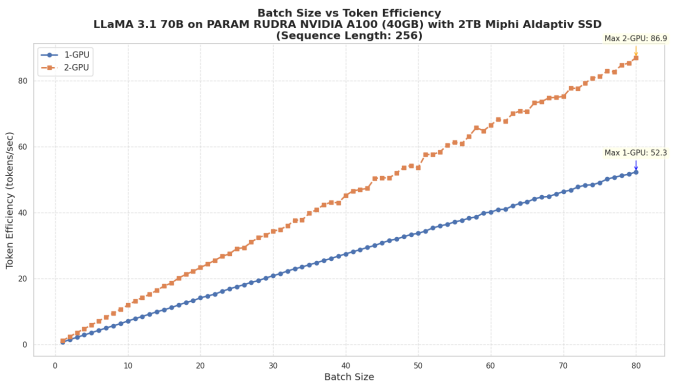


Figure 8. Batch Size vs Token Efficiency for Llama 3.1 70B on PARAM Rudra (1-GPU vs 2-GPU) with 2TB MiPhi AI SSD, Sequence Length: 256

VII. OBSERVATIONS

These observations summarize how the MiPhi aiDAPTIV AI SSD influences fine-tuning behavior across heterogeneous compute nodes, focusing on practical configuration choices, stability boundaries and operational checklists. The intent is not to rank systems, but to highlight how interconnects,

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE VI. TOKENS/SEC FOR LLAMA 2-7B AND LLAMA 3.1-8B ON RUDRA ACROSS GPU AND BATCH SIZE CONFIGURATIONS.

Max Seq.	Batch Size	Llama 2-7B		Llama 3.1-8B	
		1 GPU	2 GPU	1 GPU	2 GPU
128	8	39	84	35	77
	16	66	133	59	127
	24	85	171	75	164
	32	100	188	85	189
256	8	78	165	69	154
	16	173	259	117	256
	24	167	332	144	321
	32	196	362	171	388
512	8	153	324	137	302
	16	254	500	225	479
	24	323	629	282	600
	32	371	673	325	677
1024	8	294	611	265	587
	16	476	902	422	871
	24	593	1113	OOM	OOM
	32	679	1171	OOM	OOM
2048	8	547	1093	488	1028
	16	846	1498	OOM	OOM
	24	1000	1732	OOM	OOM
	32	1093	1703	OOM	OOM

memory capacity and storage paths shape the effective use of MiPhi, so practitioners can select batch and sequence settings, scaling strategies and checkpoint policies that deliver predictable throughput and robust runs across both dense multi-GPU and modest GPU environments.

- **Role of system topology:** Inter-GPU connectivity, host memory capacity and storage throughput primarily determine how far MiPhi aiDAPTIV can raise feasible batch size and sequence length before encountering OOM or stalls. Dense interconnects help scaling efficiency, while smaller memory footprints benefit from careful batch and sequence tuning with MiPhi's offload and prefetch mechanisms.
- **MiPhi's core value - feasibility envelope expansion:** MiPhi aiDAPTIV consistently extends the workable configuration space by offloading activations and parameters and overlapping I/O, enabling larger batches and longer contexts to fit on both dense and modest GPU nodes. Effectiveness is seen in enabling configurations that otherwise fail under baseline GPU-only fine-tuning.
- **Throughput vs. stability trade-off:** Increasing the batch size generally improves tokens/sec until memory or backend limits are reached. With MiPhi, the knee point shifts higher, but operators should validate convergence behavior since throughput gains do not substitute for hyperparameter tuning or task-specific objectives.
- **Sequence length is the dominant memory driver:** OOM errors correlate strongly with long contexts (e.g., 1024 to 2048), especially on larger models. Prioritizing sequence length control and using MiPhi to support moderate-to-high batches at shorter-to-moderate contexts yields stable, high-utilization runs on both system types.
- **Multi-GPU scaling considerations:** Scaling efficiency depends on interconnect and synchronization overheads. MiPhi helps by smoothing I/O and memory pressure, but communication topology still governs how close scaling approaches linearly. All-reduce timings and overlap should be monitored to tune GPU counts effectively in each environment.
- **Storage staging and checkpoint strategy:** Pre-staging tokenized data on the AI SSD and streaming checkpoints at appropriate cadence reduces stalls. Checkpoint frequency should be chosen to balance fault tolerance with I/O pressure, leveraging MiPhi's higher-throughput paths to keep GPU pipelines busy on both small and large nodes.
- **Safe operating regions per model size:** 7B models reliably sustain higher batch sizes across broader sequence ranges, whereas 13B and above benefit more from MiPhi to avoid OOM, but require conservative sequence lengths or graduated batch ramps — identifying per-model green zones and using MiPhi to widen them iteratively is recommended.
- **Practical scaling is sub-linear by design:** Doubling GPUs or batch size rarely doubles tokens/sec due to attention complexity, optimizer and dataloader overheads, interconnect characteristics and I/O contention. MiPhi should be used to mitigate memory and I/O bottlenecks while accepting intrinsic scaling limits, then optimizing overlap and micro-batching.
- **Validation on two contrasting nodes:** On a dense node, MiPhi's overlap and offload help sustain long-context training and higher aggregate throughput under multi-GPU scaling. On a lower-GPU node, MiPhi enables fitting larger configurations and improves single and dual-GPU usability, with attention to OOM thresholds at long contexts and larger batches.
- **Monitoring and diagnostics:** Tokens/sec, micro-batch latency, CUDA OOM events, offload bandwidth, checkpoint time and all-reduce and communications breakdowns should be tracked. Sequence length should be adjusted first for stability, then batch size for throughput and MiPhi profiling should be used to ensure offload and prefetch overlap is effective under the given interconnect and storage stack.
- **Model-choice guidance:** When time-to-train and stability are priorities under constrained GPUs, smaller models or shorter contexts with MiPhi-enabled larger batches are preferable. For larger models, MiPhi aiDAPTIV should be combined with parameter-efficient methods and conservative sequence lengths, then GPUs scaled as interconnect and memory allow.
- **Deployment guidance across heterogeneous clusters:** Topology-aware scheduling that matches workload char-

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

acteristics to node traits should be adopted. MiPhi should be used broadly to standardize training feasibility and batch and sequence profiles should be tailored per node type to achieve consistent utilization and predictable run stability across the fleet.

These results underscore the importance of infrastructure-aware training, particularly in large model scaling scenarios. A balance between batch size and sequence length must be strategically designed to avoid OOM errors and to fully utilize available GPU memory bandwidth and compute power.

VIII. Conclusion and Future Work

This study presents an infrastructure-aware evaluation of Llama 2 fine-tuning on CDAC's PARAM Rudra system, emphasizing both GPU memory constraints and storage I/O efficiency. Our experiments demonstrate that shorter sequence lengths enable higher batch sizes up to 256 for a sequence length of 128 without encountering OOM errors, aligning with the quadratic memory scaling of transformer models.

Results indicate that MiPhi-based aiDAPTIV acceleration facilitates efficient fine-tuning of Llama models (7B, 13B, 70B) on minimal GPU configurations (1 to 2x A100). This is achieved through hardware-aware scheduling and reduced memory bottlenecks, allowing sustained throughput (e.g., >3200 tokens/sec) in constrained environments. These insights support the feasibility of cost-effective LLM fine-tuning on existing HPC infrastructure, reducing reliance on large-scale GPU clusters.

Our findings highlight the importance of co-optimizing batch size, sequence length and I/O bandwidth to achieve scalable and stable performance, paving the way for inclusive and efficient LLM customization.

Future work will focus on large-scale multi-node training with DeepSpeed ZeRO-3, exploring varying batch sizes, sequence lengths, and LLM architectures (e.g., LLaMA-2 and LLaMA-3). It will also include direct comparisons with advanced SSD offloading frameworks such as ZeRO-Infinity [**10**] and MemAscend [**15**] to better evaluate aiDAPTIV+ against existing approaches. Additionally, experiments will extend to longer sequence lengths beyond 2048 tokens and analyze I/O bottlenecks on lower-bandwidth interconnects (e.g., PCIe Gen3), improving overall performance understanding and generalizability.

Additionally, integration with emerging optimization techniques such as quantized fine-tuning (e.g., QLoRA [**8**]) and unified memory strategies may further reduce resource footprints. As open-source LLMs scale in size and complexity, such system-aware evaluations will be critical for enabling cost-effective and accessible AI development across heterogeneous HPC infrastructures.

Acknowledgments

We sincerely thank C-DAC for providing access to the PARAM Rudra supercomputing platform under the National Supercomputing Mission (NSM). We also acknowledge the support of NSM, a joint initiative of the Department of Science and Technology (DST) and the Ministry of Electronics and Information Technology (MeitY), for advancing India's supercomputing capabilities and indigenous technologies. We are grateful to the C-DAC technical team for their support in enabling smooth execution of our experiments.

References

[1] Z. Chen et al., "A Survey on Large Language Models for Critical Societal Domains: Finance, Healthcare, and Law, " arXiv preprint arXiv: 2405.01769, 2024.

[2] A. Vaswani et al., "Attention Is All You Need," arXiv preprint arXiv:1706.03762, 2023.

[3] X.-K. Wu et al., "LLM Fine-Tuning: Concepts, Opportunities, and Challenges," *Big Data Cogn. Comput.*, vol. 9, p. 87, 2025. https://doi.org/10.3390/bdcc9040087

[4] MiPhi, "Cost-Effective Onsite LLM Training and Better Inferencing, " MiPhi Technologies, Product Datasheet, 2024. [Online]. Available: https://www.miphi.in/uploads/aiDAPTIV-Plus-PageFlyer.pdf

[5] National Supercomputing Mission (NSM), "PARAM Siddhi-AI, " C-DAC, Pune, India. [Online]. Available: https://nsmindia.in/infrastructure/nsm-systems/param-siddhi-ai/ [retrieved: Mar. 2026].

[6] National Supercomputing Mission (NSM), "Rudra-I: C-DAC's Indigenous Server Platform, " C-DAC, R&D Products. [Online]. Available: https: //nsmindia.in/rnd/rudra-i/ [retrieved: Mar. 2026].

[7] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, " arXiv preprint arXiv: 2403.14608, 2024.

[8] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models, " arXiv preprint arXiv: 2106.09685, 2021.

[9] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs, " arXiv preprint arXiv: 2305.14314, 2023.

[10] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning, " arXiv preprint arXiv: 2104.07857, 2021.

[11] Y. Zhao et al., "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel," arXiv preprint arXiv:2304.11277, 2023.

[12] M. Rakka, M. Fournarakis, O. Krestinskaya, J. Bazzi, K. N. Salama, F. Kurdahi, A. M. Eltawil, and M. E. Fouda, "Mixed-Precision Quantization for Language Models: Techniques and Prospects, " arXiv preprint arXiv: 2510.16805, 2025.

[13] Z. Guan, H. Huang, Y. Su, H. Huang, N. Wong, and H. Yu, "APTQ: Attention-aware Post-Training Mixed-Precision Quantization for Large Language Models, " in *Proc. 61st ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2024.

[14] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training, " in *Proc. 52nd Int. Conf. Parallel Processing (ICPP)*, New York, NY, USA: ACM, pp. 766–775, 2023.

[15] Y.-C. Liaw and S.-H. Chen, "MemAscend: System Memory Optimization for SSD-Offloaded LLM Fine-Tuning, " arXiv preprint arXiv: 2505.23254, 2026.

[16] K. Lv et al., "Full Parameter Fine-tuning for Large Language Models with Limited Resources, " arXiv preprint arXiv: 2306.09782, 2023.

[17] Xinyu Zhang, Nandan Thakur, Odunayo Ogundepo, Ehsan Kamalloo, David Alfonso-Hermelo, Xiaoguang Li, Qun Liu, Mehdi Rezagholizadeh, and Jimmy Lin. MIRACL: A Multilingual Retrieval Dataset Covering 18 Diverse Languages. *Transactions of the Association for Computational Linguistics*, 11:1114–1131, 2023. https://aclanthology.org/2023.tacl-1.63/.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Stress-Testing the Robustness of LLM-Based Causal Inference

Ankitkumar Patel*, Jigarkumar Patel†, Amit Kumar‡, Supreetha Sreeram§, and Venkatesh Kulkarni¶
*IntelliQuest Venture, USA, †University of Texas at Dallas, USA, ‡Texas A&M University–Corpus Christi, USA
§Pondicherry University, India, ¶IIT - Guwahati, India.
ankitkumar.patel179@gmail.com, jigarkumar.patel@utdallas.edu, akumar3@islander.tamucc.edu,
supreme172051@gmail.com, venkateshkulkarni2001@gmail.com

***Abstract*—Why do some of the world's most powerful Artificial Intelligence (AI) systems still struggle to reason about cause and effect? Causal discovery, the ability to discern the underlying drivers of a system, is fundamental to scientific inquiry and has traditionally relied on structured data and statistical methods. Recently, Large Language Models (LLMs) have been explored as an alternative paradigm for causal inference, leveraging their broad knowledge and reasoning capabilities. Yet, despite their linguistic fluency, LLMs often rely on surface-level pattern recognition rather than genuine causal logic. While it is known that LLMs stumble in this domain, we lack a precise map of where, why, and how significantly these failures occur. In this paper, we propose a systematic stress-test framework across six critical dimensions: model scale, causal graph complexity, prompting strategy, metadata quality, data quality, and sensitivity to uncertainty. Our findings reveal a significant "optimism bias" in LLMs; models consistently overestimate causal links, leading to low precision across the board. Furthermore, we demonstrate that traditional evaluation metrics can be deceptive: while reasoning-heavy models lead in performance, the highest scores often occur when uncertainty in inference is overlooked. Once uncertainty is integrated into the evaluation, performance drops, providing a more honest reflection of causal reliability. We also find that, while scaling model size improves outcomes, these gains are fragile; performance degrades sharply as the quality of metadata decreases. This work serves as a practical guide to the blind spots of LLMs, offering a clear-eyed assessment of when these tools can be integrated into a causal discovery pipeline and when they are likely to lead researchers astray.**

***Keywords*-Causal discovery; large language models (LLMs); uncertainty; evaluation metrics; sensitivity analysis.**

I. Introduction

Causal discovery, the task of inferring causal structure from observational data, has traditionally relied on statistical algorithms grounded in conditional independence testing and score optimization [1]. Recently, interest has grown in whether LLMs can serve as a new kind of causal reasoner by leveraging vast pretraining knowledge rather than numerical analysis.

However, early benchmarks such as CORR2CAUSE [2] revealed that LLMs (a) barely exceed random chance, (b) fail to generalize, and (c) collapse under simple surface-level perturbations, such as variable renaming. The authors argue that these models rely on lexical and positional cues rather than genuine causal reasoning. Building on this critique, Wu et al. [3] and the broader causal landscape analysis [4] showed that embedding LLM outputs as prior knowledge can undermine traditional causal discovery methods, with many reported gains attributable to engineered prompts rather than true inference. In contrast, Darvariu et al. [5] suggest that LLMs can serve as effective priors for causal graph discovery, while Ban et al. [6] demonstrate performance improvements when LLMs are used as heuristic guides within causal pipelines. Notably, the authors convinced that LLMs should play a role of a heuristic for search initialization rather than causal discovery.

Despite these insights, these previous works [2][3] share four significant methodological limitations: (a) evaluations are confined to small-scale networks, (b) metadata is treated as a binary condition (present or absent), (c) the influence of varying degrees of description richness on causal inference is not systematically characterized, and (d) the impact of various prompting strategies on causal discovery is not analyzed.

This paper addresses these gaps by proposing a systematic stress test framework on six critical dimensions; (1) model scale, (2) causal graph complexity, (3) prompting strategy, (4) metadata quality, (5) data quality, and (6) sensitivity to uncertainty. We also present a novel uncertainty-aware metric to evaluate causal inference. Our sensitive analysis across a subset of dimensions reveals that the failure modes identified by [2][3] do not scale linearly. Instead, specific failure patterns are qualitatively amplified at scale, and strategies effective for small graphs can become counterproductive as complexity grows. Notably, richer metadata improves performance on small networks. These findings provide a comprehensive map of the boundary conditions governing LLM reliability.

The rest of the paper is structured as follows. In Section II, we present the formal problem statement of causal discovery. The proposed framework for multi-dimensional sensitivity is outlined in Section III, and the proposed uncertainty-aware evaluation metric is defined in Section IV. Section V covers the performance analysis of state-of-art LLMs under the proposed framework. Finally, our findings and future work are summarized in Section VI.

II. Problem Statement

The primary goal of causal discovery is to recover the latent Directed Acyclic Graph (DAG) $G = (V, E)$ from observed variables $V = \{v_1, \ldots, v_n\}$. Accurately identifying these edges is essential for scientific explanation and intervention. While LLMs are increasingly used as heuristic priors in this domain, their reliability depends on interacting boundary conditions that remain poorly understood. We formalize this LLM-based inference as a function: $\hat{E} = \mathcal{F}(M, A, P, Q, D)$, where M denotes a model, A denotes a pair attributes, P denotes a prompt configuration, Q denotes metadata quality, and D denotes data. The fundamental challenge is to treat

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

$\mathcal{F}$ as a stable reasoner, ignoring its extreme sensitivity to perturbations in M, P, Q and D.

III. Multi-Dimensional Sensitivity Analysis

To provide a comprehensive map of LLM reliability, causal discovery can be analyzed across the following six domains.

A. Graph Scale and Topology Complexity

A causal graph's complexity is defined by its node count and edge density. As variables increase, the space of potential causal hypotheses grows combinatorially, creating two compounding challenges for LLMs. First, larger graphs result in crowded contexts, forcing the model to process overwhelming lists of relationships and descriptions. Second, finite context-length constraints risk implicit pruning, where the model deprioritized or truncates critical information. These factors limit the model toward surface heuristics rather than genuine reasoning. Consequently, LLM performance often exhibits sharp failures upon reaching capacity limits rather than degrading smoothly with topological complexity.

B. Model Scale and Pretraining Objectives

LLM effectiveness in causal discovery depends on non-linear interactions between model scale, training data, and pretraining objectives. This complexity makes it difficult to attribute performance gains solely to model size. Empirically, models optimized for reasoning subject to outperform standard LLMs on structured causal assessments. While next-token prediction models often rely on superficial lexical or frequency-based cues, true causal discovery requires specific inductive biases. Consequently, increasing model capacity only improves causal reasoning when paired with objectives that promote abstraction, counterfactual thinking, and logical consistency.

C. Prompting Strategy and Heuristic Bias

Prompting strategy centralizes LLM causal behavior, with methods like zero-shot, few-shot, and Chain-of-Thought (CoT) significantly altering performance. While CoT can elicit structured reasoning and uncover implicit assumptions through step-by-step explanation [7], its benefits are scale-dependent. As graph complexity increases, CoT prompts induce heavy context overhead, often leading to verbose but shallow reasoning. This can force models to prioritize narrative coherence over causal validity, amplifying heuristic biases. Consequently, prompting strategies that appear effective for small-scale problems may become counterproductive in large-scale problems.

D. Metadata Quality and Description Richness

Metadata refers to the semantic labels and textual descriptions of variables. While richer metadata can reduce ambiguity and provide the contextual cues necessary for causal reasoning, but on the other hand, it introduces contextual complexity which can overwhelm the reasoning capacity of LLMs. Consequently, the impact of metadata on causal accuracy depends on a non-linear trade-off between description precision, prompt structure, and the graph scale.

E. Observational Data Precision

The influence of observational data on LLM-based causal reasoning remains an open question, as interpretations are shaped by how data is represented and contextualized within a prompt rather than by numerical content alone. In practice, the granularity of observations is critical. While coarsely aggregated data may obscure causal signals, overly fine-grained representations can introduce noise and inflate prompt complexity. Consequently, LLM decision-making varies significantly based on data precision and formatting. This necessitates a careful balance between informational fidelity and contextual clarity in causal discovery tasks.

F. Impact of Evaluation Metrics

Conventional metrics like precision, recall, and F-score assume a binary framework, conflating uncertainty with error. This obscures the critical distinction between an incorrect assertion and a principled abstention, a limitation especially pronounced in LLMs, which lack classically calibrated probabilities. In our setting, we query LLMs with dual prompts for the existence ($P(e)$) and non-existence ($P(\neg e)$) of causal edges. Because LLMs rely on heuristic reasoning rather than coherent probabilistic inference, they frequently violate basic constraints (e.g., $P(e) + P(\neg e) = 1$). We also observe borderline cases where nearly equivalent probabilities reflect epistemic uncertainty rather than a meaningful causal preference. Treating these as hard decisions artificially inflates false positives and negatives. To better assess the trade-off between decisiveness and reliability, we propose a novel ternary decision framework ($True$, $False$, and $Uncertain$). By categorizing inconsistent or marginal predictions as "Uncertain," we can penalize confident errors more severely than informed abstentions. Our proposed metric is formally defined in the following section.

IV. Uncertainty-Aware Evaluation Theory

LLM-assisted causal discovery operates under inherent epistemic uncertainty, which is not adequately captured by traditional binary evaluation metrics such as precision, recall, and F-score. Unlike classical statistical methods that enforce hard decisions, LLMs naturally admit a third outcome, abstention, when confidence is insufficient. To account for this behavior, we adopt a ternary decision framework in which each ordered variable pair has a ground-truth state of *Edge* or *No-Edge*, while the model may predict *Edge*, *No-Edge*, or *Uncertain*. This yields a ternary confusion structure that distinguishes uncertainty over true edges (UE) from uncertainty over non-edges (UN), reflecting their differing implications for causal reasoning and intervention planning.

We incorporate this structure into uncertainty-adjusted precision and recall, defined as

$$R_{adj} = \frac{TP}{TP + FN + \beta \cdot UE}, \quad P_{adj} = \frac{TP}{TP + FP + \alpha \cdot UN},$$

where $\alpha, \beta \in [0, 1]$ control the penalty assigned to abstention. These parameters encode epistemic preferences between decisiveness and caution. The resulting F-Adj metric, defined as

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE I. REPRESENTATIVE WEIGHTING STRATEGIES AND THEIR EPISTEMIC INTERPRETATIONS.

(α, β)	Interpretation
$(0.5, 0.5)$	Balanced and symmetric treatment of uncertainty.
$(1.0, 1.0)$	Uncertainty treated as full failure; decisiveness is enforced.
$(0.8, 0.4)$	Abstention is preferred over guessing a causal edge.
$(0.4, 0.8)$	Guessing a causal edge is preferred over abstention.
$(0.2, 0.8)$	Aggressive discovery setting that prioritizes recall over caution.
$(0.8, 0.2)$	Conservative discovery setting that prioritizes precision and abstention.

the harmonic mean of P_{adj} and R_{adj}, generalizes the classical F-score by explicitly modeling uncertainty.

The weighting parameters (α, β) in the uncertainty-aware evaluation metric encode explicit epistemic policies that regulate the trade-off between decisiveness and caution in causal assessment. Rather than treating uncertainty as uniformly undesirable, these parameters allow evaluators to specify when a model should abstain versus when it should commit to a causal claim. Symmetric settings ($\alpha = \beta$) reflect balanced treatment of uncertainty, while asymmetric configurations express task-dependent priorities. For instance, $\alpha > \beta$ favors edge assertion by penalizing uncertainty over non-edges, whereas $\alpha < \beta$ prioritizes caution by discouraging uncertain edge claims. Such distinctions are critical in causal discovery, where the relative costs of false positives and false negatives vary across domains. By making these epistemic assumptions explicit, the framework aligns evaluation with downstream objectives and supports principled transitions between exploratory and confirmatory stages of a causal discovery pipeline. Table I summarizes representative weighting strategies and their corresponding epistemic goals.

V. PERFORMANCE EVALUATION

All experiments were conducted on the Asia network, a well-established benchmark from the Bayesian network literature [8]. The dataset models a small medical diagnostic system with eight binary variables related to pulmonary diseases and clinical symptoms. Despite its limited scale, the Asia network captures key causal motifs such as confounding and mediation, making it a canonical testbed for controlled causal discovery. The ground-truth structure is provided as a directed acyclic graph (DAG) encoding medically grounded causal relationships, including the effects of travel and smoking on disease outcomes and downstream symptoms, enabling precise evaluation of structural correctness and error types.

LLMs are evaluated under a zero-shot, instruction-based prompting setting. Specifically, we consider `GPT-4.1-mini` [9], `GPT-4.1` [10], `GPT-4o` [11], `DeepSeek-R1` [12], `GPT-5.2` [13], and `gpt-oss-20b` [14]. All models are accessed through a unified API-based evaluation pipeline and receive an identical task description along with the same rule-based instructions for identifying adjacency, spuriousness, and independence between attributes and metadata. This setup minimizes confounding effects from advanced prompt engineering, such as exemplar bias and context overload, and allows performance differences to be attributed more directly to model behavior and metadata quality. Full prompt templates and implementation details are provided in [15]; exploration of more advanced prompting strategies is left for future work.

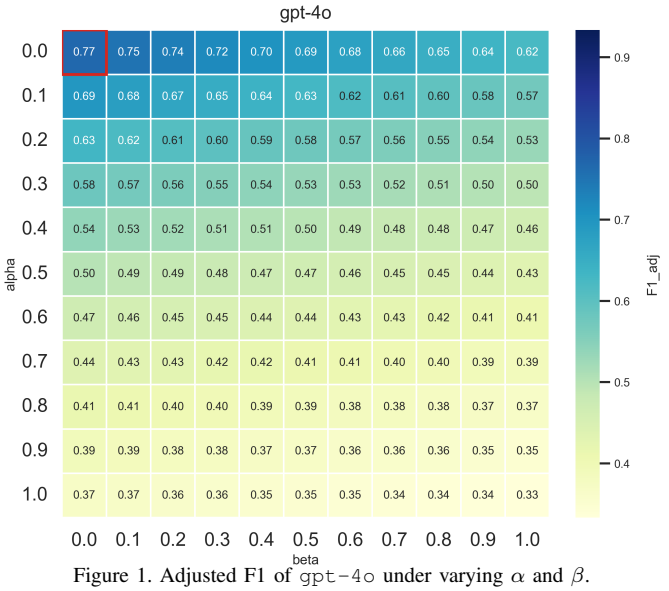


Figure 1. Adjusted F1 of `gpt-4o` under varying α and β.

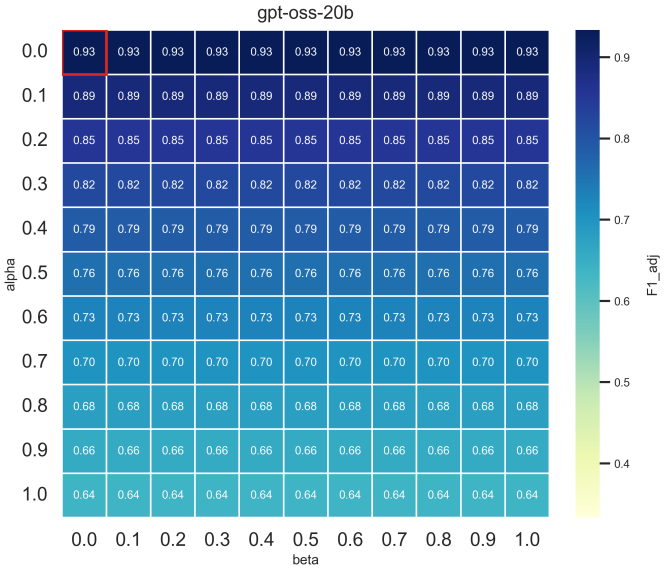


Figure 2. Adjusted F1 of `gpt-oss-20b` under varying α and β.

To examine the impacts of metadata systematically, we provided metadata with different level of richness, simulating increasing levels of noise and ambiguity. For Asia, metadata is provided at three informativeness levels (L1–L3). At L1, only variable names are provided, introducing maximum noise and forcing reliance on lexical cues. L2 augments names with variable roles and brief descriptions, reducing ambiguity without revealing explicit structure. L3 further explicit causal hints, latent variables, and confounding structure, making portions of the underlying graph approximately recoverable.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

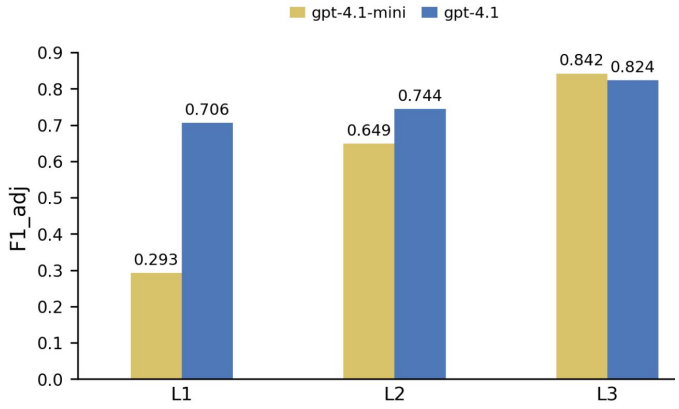


Figure 3. Adjusted F1 as a function of metadata quality on the Asia dataset.

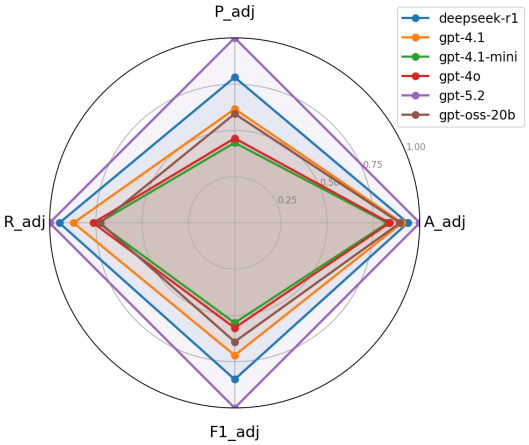


Figure 4. Comparison of evaluation metrics across models.

This controlled degradation framework enables the isolation of metadata-driven failure modes and distinguishes errors due to limited semantic grounding.

For each metadata variant, each model was queried twice using a simple vanilla prompting strategy: once with an *edge* prompt to estimate the likelihood of a directed causal relation between a variable pair, and once with a *no-edge* prompt to estimate the likelihood of no direct causal relation. In this setting, the LLM is provided only with the task description and metadata. This design choice minimizes confounding effects introduced by advanced prompt engineering, such as exemplar bias or context overload, and enables clearer attribution of performance differences to intrinsic model behavior and metadata quality. We used a fixed temperature of 0.0 and decision threshold of 0.7. For uncertainty-aware evaluation, the penalty parameters α and β were varied over $\{0.0, 0.1, \ldots, 1.0\}$, and predictions were aggregated into adjusted metrics, including `F1_adj`.

The heatmaps in Fig. 1 and Fig. 2 show how model performance ($F1_{adj}$) varies as a function of the uncertainty penalty parameters, α and β. In all models, the highest scores (max) are consistently observed in $(\alpha = 0.0, \beta = 0.0)$, where uncertainty is effectively ignored. As α and β increase, forcing the models to be penalized for ambiguous causal claims, the $F1_{adj}$ scores drop significantly. Among the models, GPT-5.2 and DeepSeek-R1 demonstrate near-perfect performance (1.00) maintaining high stability even as the penalty for uncertainty increases. GPT-4o and GPT-OSS-20B show high sensitivity to α and β. The score of GPT-4o is degraded by approximately 0.29 as decisiveness is enforced. GPT-OSS-20B exhibits the most dramatic drop, with its performance falling by 0.44 at maximum penalty. This confirms that models often appear more capable than they are by making "guesses" that a binary metric would treat as a hard decision.

In Fig. 3, the performance of gpt-4.1-mini and gpt-4.1 on the Asia dataset was evaluated using the $F1_{adj}$ metric ($\alpha = 0.5, \beta = 0.5$), which balances incorrect assertions against principled abstentions. The results reveal a non-linear relationship between metadata richness and accuracy. While both models improve as metadata moves from L1 to L3, gpt-4.1-mini shows the most dramatic gain, jumping from 0.293 to 0.842, corresponding to an improvement of approximately 187.4%. At low richness (L1), the larger gpt-4.1 significantly leads (0.706), indicating stronger internal reasoning when context is sparse. However, as metadata richness increases, the performance gap between the smaller and larger models narrows substantially. At L3, the models converge, and the smaller variant slightly leads (0.842 vs 0.824). In contrast, gpt-4.1 shows a more modest improvement of approximately 16.7% from L1 to L3, suggesting that richer metadata reduces the performance difference between smaller and larger models.

The radar chart in Fig. 4 provides a holistic view of model performance across four adjusted metrics: P_{adj}, R_{adj}, A_{adj}, and $F1_{adj}$ when $\alpha = 0.5$ and $\beta = 0.5$. Most models exhibit an "elongated" shape toward R_{adj}, indicating higher recall than precision. This provides empirical evidence for the "optimism bias", LLMs are inclined to over-identify causal links, leading to a high rate of false positives. GPT-5.2 occupies the outermost perimeter of the chart, showing nearly perfect scores (1.00) across all four metrics. DeepSeek-R1 follows closely, though it shows a slightly higher tendency toward recall over precision compared to the top frontier. Smaller models like GPT-4.1-mini and GPT-4o occupy a much smaller area of the radar, with significant dips in P_{adj}. This reflects, their tendency to rely on surface-level patterns and metadata hints rather than robust causal logic.

To improve readability and ground the quantitative results in concrete behavior, we examine representative model outputs. This qualitative inspection reveals clear and intuitive failure patterns that help explain the performance differences in Fig. 4. For instance, in the Asia network, several mid- and low-capacity models incorrectly infer a direct causal link from *smoking* to *dyspnoea*, ignoring the mediating disease variables despite metadata that explicitly describes the indirect relationship. In other cases, models assert bidirectional dependencies between clinically unrelated variables, often driven by superficial co-occurrence cues in the metadata. These errors become more frequent as metadata quality degrades or context

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

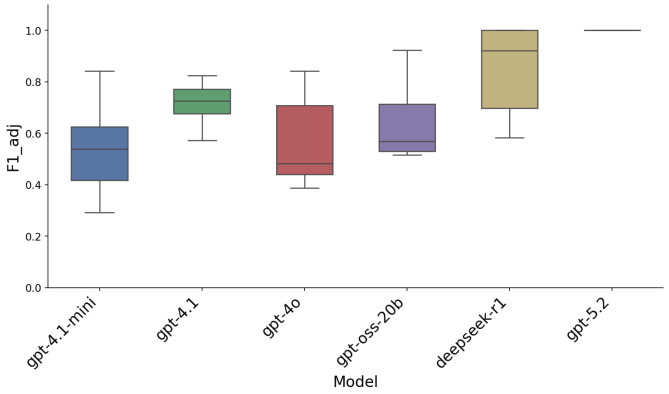


Figure 5. Distribution of adjusted F1 scores across models.

becomes crowded. In contrast, GPT-5.2 more often abstains in such ambiguous situations, which is reflected in its higher and more stable $F1_{adj}$ scores.

Figure 5 captures the variance and median performance of $F1_{adj}$ across different models in a box plot when the input context randomly varies between $L1$ to $L3$ levels. GPT-5.2, with its higher capacity and enhanced reasoning abilities, achieves substantially higher median $F1_{adj}$ scores and tighter performance distributions than smaller or non-reasoning models, whereas the remaining models exhibit wide inter-quartile ranges and multiple outliers, indicating inconsistent application of causal reasoning across varying metadata qualities.

GPT-5.2 delivers the highest and most consistent $F1_{adj}$ scores, however, it also comes at a substantially higher deployment cost which makes it impractical for all use cases. The box-plot shows that several smaller or cheaper models, such as GPT-4.1 and DeepSeek-R1, achieve reasonably stable performance under favorable conditions, suggesting they can be effective when graph size and metadata quality are controlled. The uncertainty-aware $F1_{adj}$ metric helps make these tradeoffs explicit, revealing when lower-cost models are reliable enough and when their variability poses unacceptable risk. This enables practitioners to reserve frontier models for high-stakes decisions while using less expensive models for exploratory analysis or resource-constrained settings, rather than defaulting to maximum scale by assumption.

While high-capacity models typically offer superior reliability and performance, they are not always the optimal choice when operating under stringent cost, latency, and scalability constraints. By explicitly accounting for the trade-offs between these operational metrics and raw model performance, practitioners can make informed decisions when adopting the appropriate models for specific applications. Furthermore, this balanced approach enables more efficient resource provisioning and reservation within generative AI cloud environments.

VI. Conclusion and Future Work

This paper presents a systematic stress test of LLMs for causal discovery, revealing a consistent optimism bias in which models overestimate causal relationships. To overcome the limitations of binary evaluation, we introduce the uncertainty-aware $F1_{adj}$ metric, which shows that many apparent performance gains disappear once epistemic uncertainty is accounted for. While reasoning-optimized models achieve stronger performance, smaller models like GPT-4.1-mini, narrowing the gap with larger models as metadata richness increases. Collectively, these findings provide a more realistic and reliable assessment of LLM capabilities in causal reasoning tasks. Building on these boundary conditions, future work will evaluate how LLM judgments shift when provided with raw observational data alongside variable descriptions. We plan to move beyond zero-shot prompting to study how advanced strategies, such as CoT and multi-agent debate, interact with graph scale. Finally, we will expand evaluations to complex systems exceeding 200 nodes to transform LLMs into robust components of high-stakes causal discovery pipelines.

References

[1] C. Glymour, K. Zhang, and P. Spirtes, "Review of causal discovery methods based on graphical models," *Frontiers in genetics*, vol. 10, p. 524, 2019.

[2] Z. Jin et al., "Can large language models infer causation from correlation?" In *The Twelfth International Conference on Learning Representations*, OpenReview.net, 2024. [Online]. Available: https://arxiv.org/pdf/2306.05836.

[3] X. Wu, K. Yu, J. Wu, and K. C. Tan, "LLM cannot discover causality, and should be restricted to non-decisional support in causal discovery," 2025, arXiv: 2506.00844 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2506.00844.

[4] Z. Zhang, R. Guo, Z. Wen, and Z. Zhang, "Large language models for causal discovery: Current landscape and future directions," 2024, arXiv: 2402.11068 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2402.11068.

[5] V.-A. Darvariu, S. Hailes, and M. Musolesi, "Large language models are effective priors for causal graph discovery," 2024, arXiv: 2405.13551 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.13551.

[6] T. Ban et al., "Integrating large language model for improved causal discovery," 2025, arXiv: 2306.16902 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2306.16902.

[7] J. Wei et al., "Chain of thought prompting elicits reasoning in large language models," 2022, arXiv: 2201.11903 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2201.11903.

[8] S. L. Lauritzen and D. J. Spiegelhalter, "Local computation with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society: Series B*, vol. 50, no. 2, pp. 157–224, 1988.

[9] OpenAI, *Gpt-4.1 mini model*, OpenAI API documentation. Accessed: 2026-04-12, 2025. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-4.1-mini.

[10] OpenAI, *Gpt-4.1 model*, OpenAI API documentation. Accessed: 2026-04-12, 2025. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-4.1.

[11] OpenAI, *Gpt-4o model*, OpenAI API documentation. Accessed: 2026-04-12, 2024. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-4o.

[12] DeepSeek-AI et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025. [Online]. Available: https://arxiv.org/abs/2501.12948.

[13] OpenAI, *Gpt-5.2 model*, OpenAI API documentation. Accessed: 2026-04-12, 2025. [Online]. Available: https://developers.openai.com/api/docs/models/gpt-5.2.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[14] OpenAI, *Gpt-oss-120b & gpt-oss-20b model card*, Official model card. Accessed: 2026-04-12, 2025. [Online]. Available: https://openai.com/index/gpt-oss-model-card/.

[15] A. Kumar, *Base prompt data for llm causal discovery*, https://github.com/aamitssharma07/uncertainty-aware-llm-causal-discovery-repro.git, GitHub repository, accessed: 2026-04-03, 2026.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Towards Global Multi-Cloud Strategies: Insights into AWS and Alibaba Cloud Synergy

Martin G. Zizler*, Malte Prieß†, Christoph P. Neumann*

*Department of Electrical Engineering, Media and Computer Science
Ostbayerische Technische Hochschule Amberg-Weiden, Amberg, Germany
e-mail: {m.zizler1 | c.neumann}@oth-aw.de

†Faculty of Computer Science and Electrical Engineering
Kiel University of Applied Sciences, Germany
e-mail: malte.priess@haw-kiel.de

***Abstract*—Multi-cloud strategies are increasingly adopted by modern enterprises to improve agility and resilience and to reduce vendor lock-in. Integrating workloads across providers, such as Amazon Web Services (AWS) and Alibaba Cloud, remains challenging due to interoperability and migration issues. This paper presents a comparative analysis of AWS and Alibaba Cloud, focusing on architectural, service, and policy differences affecting workload migration. Using both provider-native and open source Infrastructure-as-Code tools, we conduct an exploratory case study about the migration of Internet of Things (IoT) workloads. The results highlight key technical trade-offs and best practices for secure multi-cloud deployments, offering guidance for organizations pursuing AWS and Alibaba Cloud interoperability.**

Keywords-Cloud Computing; Multi-Cloud; AWS; Alibaba Cloud; Infrastructure-as-Code.

I. Introduction

According to Statista analysts, AWS is currently the world's leading Cloud Service Provider (CSP), while Alibaba Cloud ranks fourth worldwide [1]. In contrast, within mainland China, Alibaba Cloud holds the top position, as reported by Canalys [2]. As enterprises expand internationally, region-specific regulations and preferences drive adoption of alternative providers [3]. We specifically selected these two providers because bridging the global market leader (AWS) with the dominant provider in mainland China (Alibaba Cloud) represents a highly relevant, real-world challenge for multinational enterprises that is currently underrepresented in the literature. Multi-cloud strategies enhance agility, cost efficiency, resilience, and compliance [3]–[5], helping businesses mitigate vendor lock-in and address diverse operational needs [6]. However, integrating multiple providers introduces challenges due to differences in architectures, APIs, and services, complicating interoperability and workload portability [7][8]. Although prior research addresses general multi-cloud concepts, practical guidance for migrating workloads specifically between AWS and Alibaba Cloud remains limited. While technical hurdles, such as feature gaps in managed services, can often be solved via replatforming and custom workarounds, the overarching and arguably larger challenge lies in navigating strict, legally binding regulatory environments, including data residency and cross-border transfer restrictions.

This paper presents strategies for deploying and migrating workloads across AWS and Alibaba Cloud, focusing on technical and operational challenges. Section II outlines the state of the art; Section III describes the methodology; Section IV details the comparative analysis and deployments; Section V discusses key findings; Section VI concludes with main contributions and future directions.

II. State of the Art

Prior comparative studies have predominantly focused on AWS, Azure, and Google Cloud [9]–[11], providing quantitative and qualitative benchmarks but often excluding Alibaba Cloud. Zhang et al. [12] addressed this gap through a qualitative case study, identifying core vendor competencies and service delivery mechanisms unique to Alibaba Cloud.

The quantitative and qualitative evaluation methodologies established in these previous studies represent past successes in multi-cloud benchmarking. Our work reuses these foundational approaches but extends them to practical, real-world migration scenarios and technical interoperability involving Alibaba Cloud, which remains underrepresented.

III. Methods

This section details the methodological framework used to analyze, design, and empirically validate a multi-cloud strategy across AWS and Alibaba Cloud. Our approach integrates structured comparative analysis with an exploratory case study, explicitly addressing gaps identified in prior studies and leveraging insights from recent empirical research.

A. Research Design

We employed a mixed-method comparative and exploratory approach, as advocated for cloud provider evaluations [9]. Our methodology combines a targeted literature review to identify technical, operational, and architectural challenges in multi-cloud migration with practical experimentation to ensure findings are empirically grounded.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Copyright (c) IARIA, 2026. ISBN: 978-1-68558-373-6

B. Comparative Framework

Building on the foundational approaches discussed in Section II, our comparative framework evaluates real-world migration scenarios and technical interoperability, emphasizing four domains: Global Infrastructure, Core Service Portfolio, API Usage, and Infrastructure-as-Code (IaC) tooling.

C. Strategy Development

The comparative insights provided the basis for developing a multi-cloud architectural strategy. Guided by reference architectures in the literature [13, pp. 72–76], we evaluated managed Virtual Machine (VM), container, and serverless models. Reflecting recent empirical work, such as Rajendran et al. [14], which underscores the importance of use-case-driven benchmarking, we selected a representative IoT workload for our Proof of Concept (PoC). A serverless-first strategy was adopted, supplemented by VMs where feature parity was lacking. This means that our approach is more akin to a replatforming approach rather than a simple rehosting or "lift and shift" approach [15]. Replatforming typically requires a higher technical complexity, which means that it can surface deeper migration complexities, involving a higher amount of managed services. To systematically assess migration overhead and feature coverage, we implemented both provider-native—Amazon Web Services Cloud Development Kit (AWS CDK) and Resource Orchestration Service Cloud Development Kit (ROS CDK)—and provider-agnostic (CDK for Terraform) Infrastructure-as-Code (IaC) tools.

D. Proof-of-Concept Development

To test our strategy, we designed and deployed a reference IoT application on AWS using AWS CDK, then migrated and adapted it for Alibaba Cloud with ROS CDK. Parallel definitions using CDK for Terraform provided an agnostic baseline for comparison. The implementation process, informed by best practices in IaC-driven migration [16][17], included:

- Defining and mapping equivalent resources and deployment steps for each provider,
- Adapting configurations and documenting feature gaps,
- Recording manual interventions required for successful migration.

E. Evaluation Methodology

We evaluated each deployment approach using both quantitative and qualitative criteria:

- Portability: Ease of migrating workload definitions and configurations
- Operational Transparency: Ongoing management and troubleshooting
- Maintenance Effort: Codebase maintenance
- Performance: Where measurable, indicative metrics were collected
- Security: Aligning IAM/RAM policies and Authentication

All findings were recorded systematically, with special attention to points of friction and required workarounds, as recommended by prior multi-cloud migration studies [18][19].

IV. Results

This section presents the outcomes of the systematic comparative analysis between AWS and Alibaba Cloud in Section IV-A and the exploratory case study in Section IV-B. The comparative analysis is based on vendor documentation and migration guides, which may introduce bias. To minimize overreliance on these secondary sources, we implemented the practical migration of a representative workload using IaC approaches.

A. Comparative Analysis

1) Global Infrastructure: AWS maintains global reach with 36 regions and 114 availability zones as of mid-2025, delivering strong coverage in North America and Europe [20]. Alibaba Cloud operates 29 regions and 87 availability zones, with its core strength in Greater China [21]. Both providers offer specialized partitions like AWS GovCloud or AWS China [22] to accommodate regulatory or sovereignty requirements.

It is important to note that Alibaba Cloud maintains two distinct infrastructures: AlibabaCloud.com, which serves international regions (e.g., Singapore, Frankfurt, Silicon Valley), and Aliyun.com (e.g., Shanghai, Beijing, Hangzhou), which serves regions within mainland China. In compliance with Chinese regulatory requirements, both Alibaba platforms operate in isolation. While international users can provision or manage resources in mainland China regions through AlibabaCloud.com, they are subject to a different regulatory framework (see also table II, regarding "Cross-border transfer restrictions", "Provider restrictions", and "Data residency").

2) Core Service Portfolio: Both AWS and Alibaba Cloud offer comparable core services across compute, storage, databases, networking, and security, though feature parity is not universal. Figure 1 provides an overview of the matched services and their functional completeness, based on vendor documentation [23][24] and a direct comparison by Alibaba [25]. Key differences are outlined below.

Messaging & API Management: Amazon Simple Queue Service (SQS) offers durable queuing with Standard (at-least-once) and FIFO (First-In-First-Out) modes, including dead-letter queues, short/long polling, and up to 14-day retention. Amazon Simple Notification Service (SNS) enables pub/sub delivery to SQS, Lambda, HTTP, Email, Mobile devices, SMS, Kinesis Data Firehose, and external providers (e.g., MongoDB). Alibaba Cloud Simple Message Queue (SMQ) supports queue-based and topic-based messaging with dead-letter queues, polling, and up to 7-day retention, but lacks FIFO mode. Its topic-based mode supports delivery to SMQ, Function Compute, HTTP, Email, SMS, and mobile endpoints. Both AWS API Gateway and Alibaba Cloud API Gateway are fully managed services that enable secure, scalable client-to-backend communication.

Compute: AWS EC2 and Alibaba Cloud ECS provide flexible VM types. AWS Lambda and Alibaba Function Compute offer serverless, event-driven compute with auto-scaling and pay-per-use pricing. AWS EKS and Alibaba Cloud

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

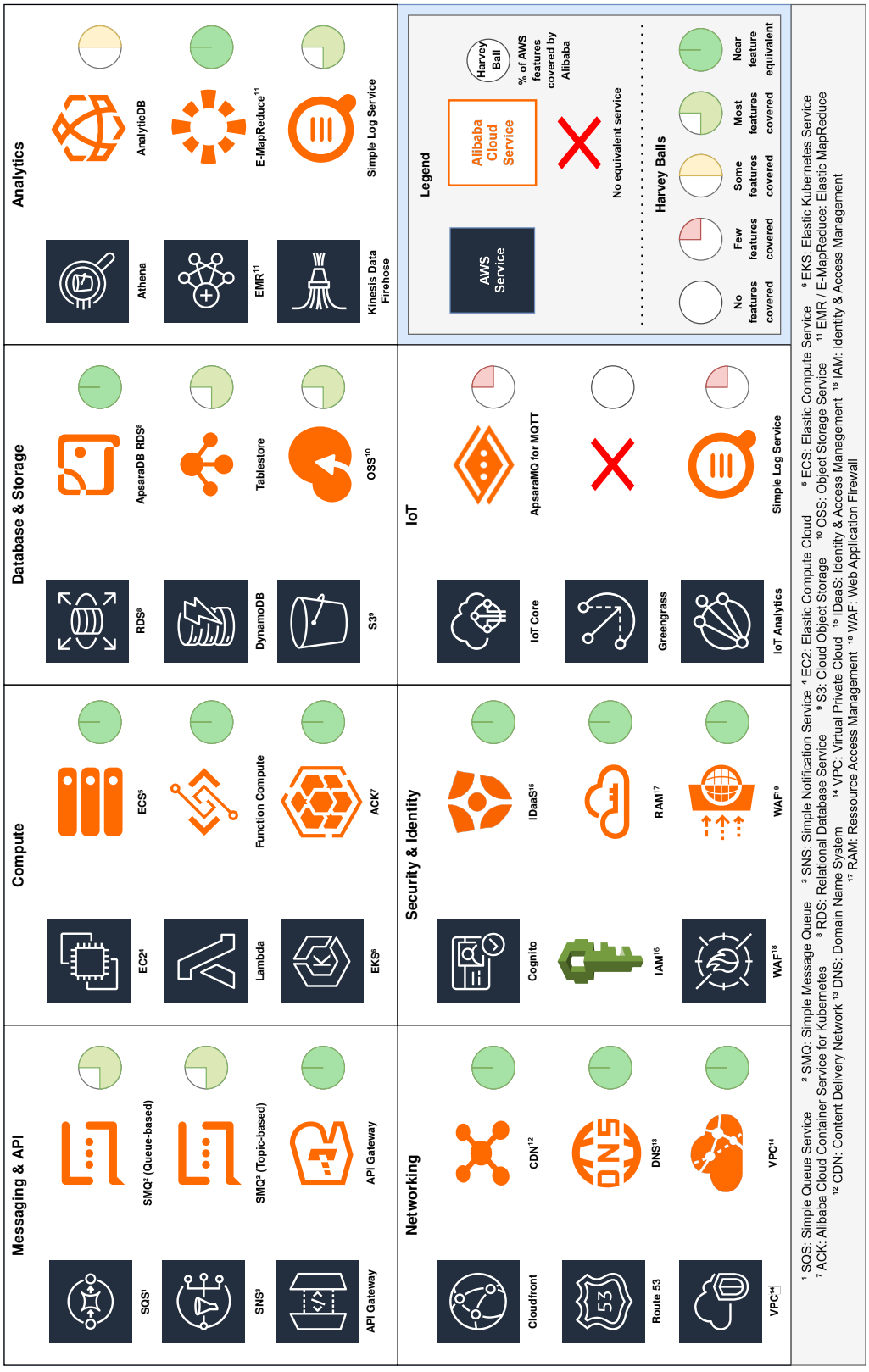


Figure 1. Cloud service overview and comparison between AWS and Alibaba Cloud core offerings, with a substantial gap in the IoT domain.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

ACK deliver managed Kubernetes with high availability and reduced operational overhead.

Database & Storage: AWS Relational Database Service (RDS) and Alibaba Cloud ApsaraDB RDS can manage SQL databases. DynamoDB is a managed, serverless NoSQL database. Table Store delivers comparable features but has no true on-demand capacity mode, as it bills per Compute Unit (CU) instead of per request. AWS S3 and Alibaba Cloud Object Storage Service (OSS) are both fully managed object-storage services. S3 leads in the number of Storage classes.

Analytics: AWS Athena is serverless and lets you run SQL directly on S3 data for fast, ad-hoc analytics with no setup. Alibaba Cloud AnalyticDB provides batch processing and real-time analysis with support for both internal data and OSS, but requires cluster configuration. Athena is simpler to use, while AnalyticDB is more complex. AWS EMR and Alibaba E-MapReduce both run managed Hadoop and Spark clusters for big data processing in the cloud. AWS Kinesis Data Firehose and Alibaba Cloud Simple Log Service (SLS) both handle real-time data ingestion, transformation, and delivery to their cloud platforms. A key difference is that Firehose focuses on streaming data delivery, while SLS also includes built-in log analytics and monitoring features.

Networking: AWS CloudFront and Alibaba Cloud CDN both accelerate web content delivery via edge caching, reducing latency and improving performance. AWS Route 53 and Alibaba Cloud DNS provide scalable, globally distributed DNS management. Both clouds support secure, isolated virtual networks through their Virtual Private Cloud (VPC) services.

Security & Identity: AWS Cognito and Alibaba Cloud IDaaS both provide cloud-based user authentication and access management, integrating with their respective cloud services. AWS Identity and Access Management (IAM) and Alibaba Cloud Resource Access Management (RAM) offer the same core features of access management, including user, group, and role management, as well as permission controls. AWS Web Application Firewall (WAF) and Alibaba Cloud WAF both protect web apps from threats like SQL injection and XSS, offering customizable rules and real-time monitoring.

IoT: AWS IoT Core supports secure device connectivity, flexible protocols, and seamless integration with other AWS services. Alibaba Cloud ApsaraMQ for MQTT provides scalable MQTT messaging but lacks advanced device management and integration features found in IoT Core. AWS Greengrass offers edge computing for IoT, enabling local compute and sync when offline. Alibaba Cloud has no direct equivalent service to Greengrass. AWS IoT Analytics delivers managed pipelines for processing IoT data, while Alibaba Cloud lacks a truly equivalent service. Simple Log Service (SLS) can be used for basic data ingestion and analytics.

3) API Usage: Both clouds expose RESTful APIs and SDKs covering major languages, but slightly differ in endpoint conventions and authentication depending on configuration. The API documentations for both CSPs show that basic API requests are still very similar across both (e.g., Bucket API documentation for Amazon S3 [26] vs. Alibaba OSS [27]).

4) Infrastructure-as-Code Tools: AWS CDK (CloudFormation) and Alibaba ROS CDK provide native IaC tooling. Meanwhile, Terraform or OpenTofu, as well as CDK for Terraform, which are popular for multi-cloud deployments, also support both CSPs [16]. Native IaC tooling generally provides faster support for new resource types and higher levels of abstraction.

B. Exploratory Case Study

1) Workload and Architecture: As a Proof-of-Concept (PoC), a representative IoT workload consisting of compute, storage, and event-driven processing was implemented using both provider-native and agnostic IaC tools for deployment. The workload includes serverless functions, object storage buckets, event triggers, and messaging services, which can be seen in Figure 2. Equivalent resources were used for AWS and Alibaba Cloud. Additional adaptation was required for the AWS IoT Core. While ApsaraMQ for MQTT exists as a potential replacement, it is just a generic MQTT broker with a very sparse feature set (see IV-A). Therefore, Thingsboard was selected as an open alternative and deployed on Alibaba Cloud ECS. Similar to IoT Core, Thingsboard fully supports X.509 Certificate-based mutual authentication, which can be managed by device [28]. It also supports custom Rule Chains to process events. To securely send data from a Thingsboard Rule Chain to other Alibaba Cloud services, a simple Flask server that can get access by utilizing the Alibaba Cloud SDK for Python was also added as an intermediary. Furthermore, an external adapter was set up to show that data can also be retrieved from the CSPs. This adapter was also used to test latency differences depending on deployment location.

2) IaC Implementation: AWS CDK (Python) and Alibaba ROS CDK (Python) were used to define and deploy the stack natively. Most of the resource definitions translated with little adaptation needed, as they have very high overlap, as shown in IV-A. The remaining required manual adaptation is due to differences in parameterization, IAM/RAM policy syntax, or missing features (e.g., managed IoT services).

CDK for Terraform (Python) was used to define stacks utilizing the same Python environment, targeting both AWS and Alibaba Cloud providers, while OpenTofu was used as the underlying IaC tool (open-source Terraform fork). Very similar to the provider-native approach, the Alibaba Cloud code base required provider-specific adaptation in all resources to accommodate differences in event sources and IAM/RAM, as well as configuration.

3) Deployment and Operational Metrics: For these metrics, the Alibaba Cloud equivalent of AWS IoT Core (an ECS-based Thingsboard deployment) is excluded from the IaC line count, as automating this setup would require substantial custom scripting. For fair comparison, AWS IoT Core is also omitted.

Table I summarizes deployment and teardown times, as well as Lines of Code (LoC) required for IaC definitions of each approach. Deployment times were similar between provider-native and provider-agnostic tools within each platform, but Alibaba Cloud was faster. Line counts were measured using

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

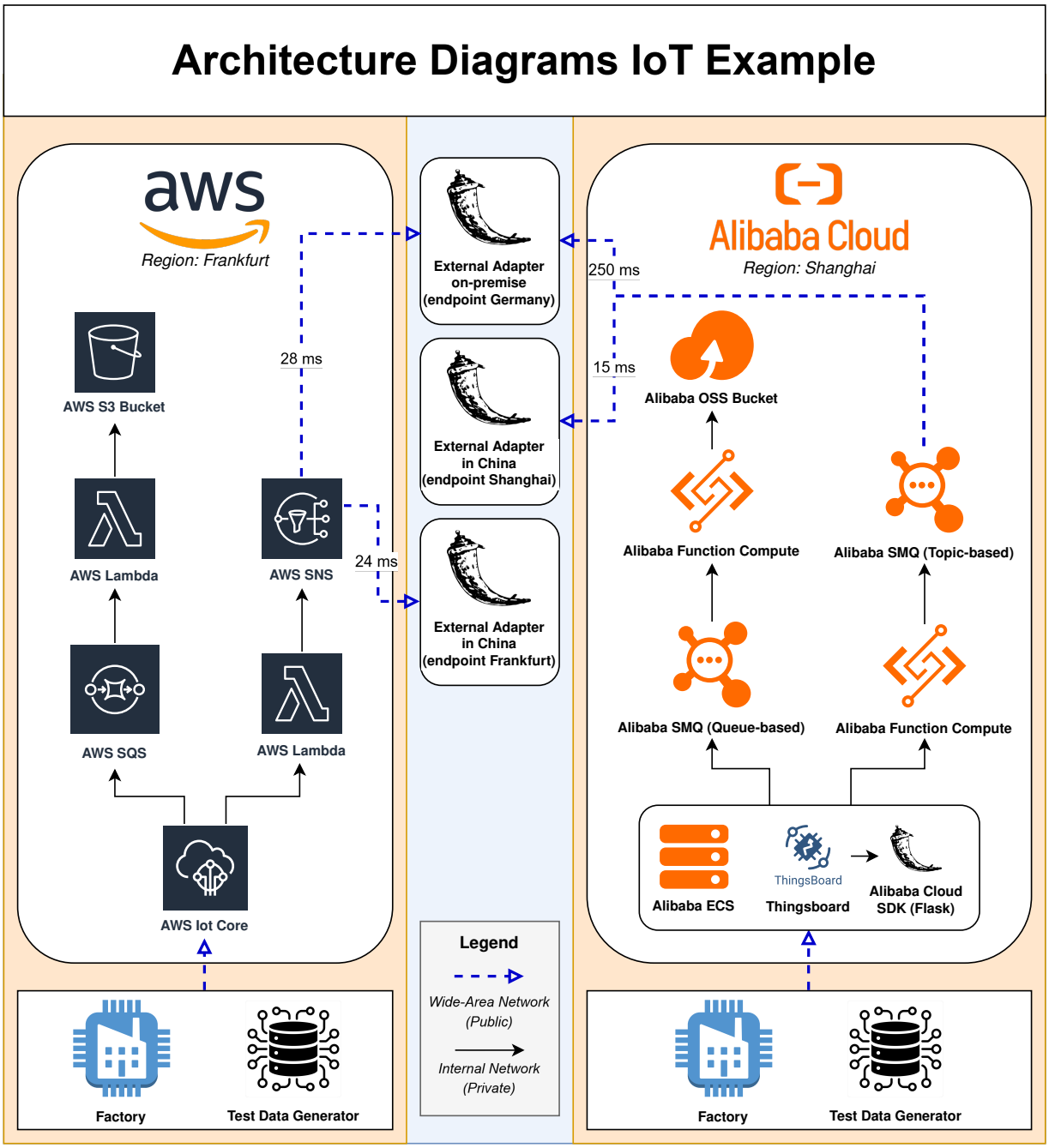


Figure 2. Architecture diagram showing resource mapping and data flow of a basic IoT Stack.

VS Code Counter with code formatted via Black to ensure consistency. Notably, the AWS CDK for Terraform (CDKTF) implementation required substantially more code than alternative approaches, primarily due to AWS's detailed IAM model and the need for explicit resource linking. In contrast, the AWS CDK benefits from high-level constructs, resulting in a more concise codebase. Differences in code length among Alibaba Cloud tools were comparatively minor.

TABLE I. DEPLOYMENT TIMES, TEARDOWN TIMES, AND IAC LINE COUNTS OF IOT STACK.

Platform	Tool	Deploy	Destroy	Lines
AWS	CloudFormation	1m 15s	56s	75
AWS	CDKTF	1m 11s	30s	184
Alibaba Cloud	ROS	24s	26s	146
Alibaba Cloud	CDKTF	19s	26s	143

The architecture diagram in Figure 2 also provides some latency numbers, annotated to the blue dashed arrows for wide-area networks. Network transfers within the same country (Germany to Germany or China to China) have low latency.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Cross-border egress (from Alibaba in Shanghai to the external adapter in Germany) introduces a latency about ten times higher. We provide an extended set of quantitative benchmarking in a publicly available Master's thesis [29].

V. Evaluation

This section summarizes the main findings from our comparative analysis and exploratory case study, highlighting key trade-offs of native versus agnostic IaC approaches.

A. Interpreting the Comparative Analysis

AWS and Alibaba Cloud both offer mature core services, but differ in regional coverage, service completeness, and compliance. IoT-heavy workloads on Alibaba Cloud require additional custom or third-party solutions to address service gaps, whereas AWS provides more integrated support.

B. Evaluating the PoC

The deployments allow evaluation of the following criteria:

1) **Portability**: Provider-native IaC (AWS CDK, ROS CDK) offers rapid access to new features and high-level constructs, but poor cross-provider code reuse. Agnostic IaC (OpenTofu) enables a unified code base, but still needs extensive provider-specific adjustments.
2) **Operational Transparency**: Native tools integrate better with CSP management interfaces, offering richer diagnostics and control, unlike OpenTofu-based stacks.
3) **Maintenance Effort**: Unified CDK for Terraform code bases can reduce duplication but increase maintenance for provider plugin updates. Native stacks benefit from vendor-managed updates but require managing separate pipelines.
4) **Performance**: Deployment times for Alibaba Cloud seem to be a bit faster (see Table I).
5) **Security**: Ensuring least-privilege access required manual effort to align IAM (AWS) and RAM (Alibaba Cloud) policies. For instance, AWS CDK provides high-level abstractions for granting permissions (e.g., allowing Lambda to write to S3), whereas ROS CDK often necessitates explicit role and policy configuration. Additionally, synchronizing certificate-based authentication across providers involved adapting identity management to maintain secure communication over wide-area networks.

C. Compliance Considerations

Compliance challenges are increased by regional regulations. For example, Alibaba Cloud's mainland China partition is subject to local laws like the China Cybersecurity Law, requiring data residency and stricter controls on cross-border flows [46]. AWS China and Alibaba's specialized regions address sovereignty but require careful architectural planning.

To further classify the regulatory requirements, table II shows examples of compliance constraints derived from international legal sources and industry-specific standards. The analysis does not aim to provide a concluding legal evaluation of regulatory frameworks; rather, it offers a first technical abstraction of selected requirements. From an architectural perspective, it illustrates how regulatory requirements, such as the GDPR, the German IT Security Act 2.0, PIPL, or the CLOUD Act, may translate into concrete technical design decisions, including role-based access control, client separation, and data localization.

Due to the limited harmonization of international regulations, globally uniform cross-country infrastructures remain challenging. A more realistic horizon lies in compliance-aware, modular architectures that enable controlled interoperability while respecting regional legal constraints.

D. Lessons Learned & Best Practices

- Utilize a service mapping matrix to track equivalences.
- Use provider-native IaC services to make use of high-level abstraction and have a higher operational transparency. Use cloud-agnostic IaC to achieve more equal code bases between different CSPs.
- Plan for manual adaptation where services don't match.
- Leverage native security tools and audit access policies.

E. Threats to Validity

Our PoC focused on a basic IoT stack; results may not generalize to large-scale data processing, or CSP-specific managed services outside the evaluated scope. Pricing and performance data are indicative; real-world figures will vary by workload size, region, and time. Finally, CSP feature sets evolve rapidly, so this mid-2025 snapshot may differ from future states.

VI. Conclusion and Future Work

Our analysis compared AWS and Alibaba Cloud across infrastructure, services, Infrastructure-as-Code tools, and regulatory frameworks, with the findings validated through a small-scale IoT proof of concept. AWS has a more mature service portfolio and leads in innovation speed. This can make it harder to develop a true multi-cloud strategy based on using serverless services. Our comparative analysis extends previous studies to real-world migration scenarios and technical interoperability. In conclusion, the paper closes several gaps in multi-cloud literature for global approaches that comprise China and Alibaba Cloud.

Several opportunities for further investigation present themselves moving forward:

- Broader workloads with other managed services.
- Performance and cost benchmarking at scale.
- Explore integration of multi-cloud management platforms.
- Assess interoperability with third-party SaaS offerings.
- Evaluate specific privacy and security implications of using CSPs governed by distinct national legal frameworks (e.g., data sovereignty and state access concerns regarding Alibaba Cloud).

By advancing these areas, future work can further reduce operational friction and enhance the robustness of global multi-cloud deployments.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE II. EXAMPLES OF COMPLIANCE CONSTRAINTS DERIVED FROM INTERNATIONAL LEGAL SOURCES AND INDUSTRY-SPECIFIC STANDARDS.

Constraint type		Source	Significance for multi-cloud
Data residency	Transfer of personal data to third countries (outside the EU/EEA) is only permitted under specific conditions.	GDPR Art. 44–49 [30]	Storing EU personal data in Alibaba Cloud Mainland may breach GDPR; transfers require adequacy decisions, SCCs, or legal exceptions.
Cross-border transfer restrictions	Data exports from China may require prior security assessment and government approval.	DSL Art. 31–37 [31], CSL Art. 37 [32], MLPS 2.0 [33]	Transfers from Alibaba Cloud Mainland to AWS Frankfurt may require CAC approval and data export security review.
Tenant isolation	Data belonging to different customers, departments, or patients must be kept logically and technically separated.	GoBD 2020 [34], SOX §404 [35], HIPAA 164.308(a)(4) [36]	IaC should enforce resource separation (e.g., VPCs, IAM roles, storage buckets) per tenant to prevent data leakage.
Auditability	Access to systems must be traceable and securely logged for compliance and incident analysis.	BSI C5:2020 (e.g., OPS-07) [37], ISO 27001 8.15 (Logging) [38], GDPR Art. 30, 33 [30]	Cloud-native logging (e.g., AWS CloudTrail, Alibaba ActionTrail) should be enabled, retained, and protected.
Classification requirements	Operators of critical infrastructure must classify systems and apply tiered protection accordingly.	NIS 2 (EU 2022/2555, Art. 21) [39], BSIG §8a [40] & IT Security Act [41], MLPS 2.0 [33]	Selection of certified services only (e.g., BSI C5) and onshore deployment; additional monitoring and emergency mechanisms if necessary; classification as a necessary prerequisite for protective measures.
Provider restrictions	A cloud provider may be subject to foreign government access demands (e.g., US CLOUD Act, CN national laws).	CLOUD Act [42], GDPR Art. 48 [30], BSIG §9b [40] & IT Security Act [41], Gaia-X standards, if applicable [43][44]	Onshore providers preferred to avoid extraterritorial access; regulatory context may disqualify US/CN providers for critical workloads; clarification on who can enforce access to data is essential.
Access and identity control	Only authorized users should access data, using strong authentication and role-based access control.	GDPR Art. 32(1)(b) [30], ISO 27001 8 (Technological controls) [38], BSI C5 (e.g., IDM-09) [37], CSL Art. 21 [32]	IAM (AWS) and RAM (Alibaba) should enforce RBAC, MFA, and auditable access policies.
Data minimization & purpose limitation	Only necessary data may be processed and stored for clearly defined purposes.	GDPR Art. 5(1)(c) [30], PIPL Art. 6 [45]	IaC and pipelines should be limited to minimal datasets and clearly scoped processing goals.

REFERENCES

[1] F. Richter, "Amazon and Microsoft stay ahead in global cloud market," 2025, Accessed: 2025-03-18. [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

[2] D. Singh, "Canalys: Global Cloud Infrastructure Spending Rose 22% In Q2 2025," 2025, Accessed: 2025-09-11. [Online]. Available: https://channelpostmea.com/2025/09/11/canalys-global-cloud-infrastructure-spending-rose-22-in-q2-2025/.

[3] D. Seth, M. Najana, and P. Ranjan, "Compliance and regulatory challenges in cloud computing: A sector-wise analysis," *International Journal of Global Innovations and Solutions (IJGIS)*, vol. 3, Jun. 2024, https://ijgis.pubpub.org/pub/n5sgt1c7. DOI: 10.21428/e90189c8.68b5dea5.

[4] J. Alonso et al., "Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review," *Journal of Cloud Computing*, vol. 12, p. 6, Jan. 2023. DOI: 10.1186/s13677-022-00367-6.

[5] G. Chatzithanasis, E. Filiopoulou, C. Michalakelis, and M. Nikolaidou, "Exploring cost-efficient bundling in a multi-cloud environment," *Simulation Modelling Practice and Theory*, vol. 111, p. 102 338, May 2021. DOI: 10.1016/j.simpat.2021.102338.

[6] D. Petcu, "Multi-cloud: Expectations and current approaches," in *Proceedings of the 2013 International Workshop on Multi-Cloud Applications and Federated Clouds*, ser. MultiCloud '13, Prague, Czech Republic: Association for Computing Machinery, 2013, pp. 1–6. DOI: 10.1145/2462326.2462328.

[7] R. Ranjan, "The cloud interoperability challenge," *IEEE Cloud Computing*, vol. 1, no. 2, pp. 20–24, 2014. DOI: 10.1109/MCC.2014.41.

[8] V. Munteanu, C. Sandru, and D. Petcu, "Multi-cloud resource management: Cloud service interfacing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 3, p. 3, Dec. 2014. DOI: 10.1186/2192-113X-3-3.

[9] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, Melbourne, Australia: Association for Computing Machinery, 2010, pp. 1–14. DOI: 10.1145/1879141.1879143.

[10] M. Saraswat and R. Tripathi, "Cloud computing: Comparison and analysis of cloud service providers – AWS, Microsoft and Google," in *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, 2020, pp. 281–285. DOI: 10.1109/SMART50582.2020.9337100.

[11] V. V. Rajendran and S. Swamynathan, "Parameters for comparing cloud service providers: A comprehensive analysis," in *2016 International Conference on Communication and Electronics Systems (ICCES)*, 2016, pp. 1–5. DOI: 10.1109/CESYS.2016.7889826.

[12] G. Zhang and M. Ravishankar, "Exploring vendor capabilities in the cloud environment: A case study of Alibaba cloud computing," *Inf. Manage.*, vol. 56, no. 3, pp. 343–355, Apr. 2019. DOI: 10.1016/j.im.2018.07.008.

[13] J. Mulder, *Multi-Cloud Administration Guide: Manage and Optimize Cloud Resources Across Azure, AWS, GCP, and Alibaba Cloud*. De Gruyter, 2024, ISBN: 9781501519482.

[14] P. Rajendran, S. Maloo, R. Mitra, A. Chanchal, and R. Aburukba, "Comparison of cloud-computing providers for deployment of object-detection deep learning models," *Applied Sciences*, vol. 13, p. 12 577, Nov. 2023. DOI: 10.3390/app132312577.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[15] M. Hussain, "A comparative analysis of cloud migration strategies for enterprise systems architecture," *World Journal of Advanced Engineering Technology and Sciences*, vol. 15, pp. 747–756, May 2025. DOI: 10.30574/wjaets.2025.15.2.0622.

[16] R. Kyadasu, "Exploring infrastructure as code using Terraform in multi-cloud deployments," *SSRN Electronic Journal*, Jan. 2025. DOI: 10.2139/ssrn.5075647.

[17] S. Achar, "Enterprise SaaS workloads on new-generation Infrastructure-as-Code (IaC) on multi-cloud platforms," *Global Disclosure of Economics and Business*, vol. 10, pp. 55–74, Jul. 2021. DOI: 10.18034/gdeb.v10i2.652.

[18] H. Zhao, Z. Benomar, T. Pfandzelter, and N. Georgantas, "Supporting multi-cloud in serverless computing," in *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, 2022, pp. 285–290. DOI: 10.1109/UCC56403.2022.00051.

[19] V. Yussupov, U. Breitenbücher, F. Leymann, and C. Müller, "Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends," Dec. 2019, pp. 273–283. DOI: 10.1145/3344341.3368813.

[20] Amazon, "AWS Infrastructure," 2025, Accessed: 2025-05-08. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.

[21] Alibaba, "Alibaba Cloud Infrastructure," 2025, Accessed: 2025-05-08. [Online]. Available: https://www.alibabacloud.com/en/global-locations?_p_lc=1#J_5253092060.

[22] Amazon, "AWS China," 2025, Accessed: 2025-05-08. [Online]. Available: https://www.amazonaws.cn/en/about-aws/china/.

[23] Amazon, "AWS Documentation," 2025, Accessed: 2025-03-23. [Online]. Available: https://docs.aws.amazon.com/.

[24] Alibaba, "Alibaba Cloud documentation," 2025, Accessed: 2025-03-23. [Online]. Available: https://www.alibabacloud.com/help/en.

[25] Alibaba, "Alibaba Cloud Services Migration Guide Service Comparison," Accessed: 2025-08-04. [Online]. Available: https://www.alibabacloud.com/en/product/product-mapping.

[26] Amazon, "AWS S3 API Documentation," 2025, Accessed: 2025-05-05. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/API/API_ListObjectVersions.html.

[27] Alibaba, "Alibaba OSS API Cloud Documentation," 2025, Accessed: 2025-05-05. [Online]. Available: https://www.alibabacloud.com/help/en/oss/developer-reference/listobjectversions?spm=a2c63.p38356.0.i2#reference-n2s-xy3-fhb.

[28] Thingsboard, "Thingsboard Documentation," Accessed: 2025-08-04. [Online]. Available: https://thingsboard.io/docs.

[29] M. Zizler, "Global Multi-Cloud Strategies: Efficient Utilization of AWS and Alibaba Cloud for Scalable Cloud Applications," Master's Thesis, Ostbayerische Technische Hochschule Amberg-Weiden, Sep. 2025. DOI: 10.5281/zenodo.17400896.

[30] European Union, "General Data Protection Regulation (GDPR), Regulation (EU) 2016/679," 2016, Official Journal of the European Union, L 119, 4 May 2016.

[31] National People's Congress of the People's Republic of China, "Data Security Law of the People's Republic of China (DSL)," 2021, adopted on 10 June 2021, effective from 1 September 2021, translation by DigiChina (Stanford University).

[32] National People's Congress of the People's Republic of China, "Cybersecurity Law of the People's Republic of China (CSL)," 2017, adopted on 7 November 2016, effective from 1 June 2017, translation by DigiChina (Stanford University).

[33] Ministry of Public Security of the People's Republic of China, "Multi-Level Protection Scheme 2.0 (MLPS 2.0) – Classified Protection of Cybersecurity," 2019, adopted on 13 May 2019, effective 1 December 2019, translation not publicly available.

[34] Federal Ministry of Finance, "Principles for the proper management and storage of books, records and documents in electronic form and for data access," 2019, translation, notice dated 28 November 2019, valid from 1 January 2020.

[35] United States Congress, "Sarbanes–Oxley Act (SOX)," 2002, Public Law 107–204, enacted July 30, 2002.

[36] U.S. Department of Health and Human Services, "Health Insurance Portability and Accountability Act (HIPAA) of 1996, Security Rule – 45 CFR §164.308(a)(4) Information Access Management," 1996, codified in Title 45 of the Code of Federal Regulations (CFR), Subpart C – Security Standards for the Protection of Electronic Protected Health Information.

[37] Federal Office for Information Security (BSI), "Cloud Computing Compliance Controls Catalogue (C5:2020)," 2020.

[38] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), "ISO/IEC 27001:2022 – Information security, cybersecurity and privacy protection – Security techniques – Information security management systems – Requirements," 2022.

[39] European Parliament and Council of the European Union, "Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union (NIS 2 Directive)," 2022, Official Journal of the European Union, L 333, 27 December 2022, pp. 80–152.

[40] Federal Republic of Germany, "Act on the Federal Office for Information Security (BSI Act – BSIG)," 2021, translation, as amended by the IT Security Act 2.0 of 28 May 2021.

[41] Federal Republic of Germany, "Second Act on increasing the Security of IT Systems (German IT Security Act 2.0)," 2021, translated, promulgated in the Federal Law Gazette I, No. 25, 27 May 2021, pp. 1086–1103.

[42] United States Congress, "Clarifying Lawful Overseas Use of Data Act (CLOUD Act), H.R. 4943," 2018, enacted March 23, 2018, as Division V of the Consolidated Appropriations Act, 2018 (Public Law 115–141).

[43] Gaia-X European Association for Data and Cloud, "Gaia-x policy rules document," 2022, Version 22.04, April 2022.

[44] Gaia-X European Association for Data and Cloud, "Gaia-x trust framework," 2022, Version 22.10, October 2022.

[45] National People's Congress Standing Committee, "Personal Information Protection Law of the People's Republic of China (PIPL)," 2021, adopted on 20 August 2021, effective 1 November 2021, translation by DigiChina (Stanford University).

[46] Cyberspace Administration of China, "Questions and Answers on Data Outbound Security Management Policy," Apr. 2025, Accessed: 2025-07-21. [Online]. Available: https://www.cac.gov.cn/2025-04/09/c_1745906286623776.htm.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

A Multi-Resource Power Modeling Framework for Energy-Aware Cloud Simulation in CloudSim

Awet Teklemariam Ghebrekidan and Raju Shrestha

Department of Computer Science, Oslo Metropolitan University (OsloMet)
Oslo, Norway
Email:{s362104; raju.shrestha}@oslomet.no

***Abstract*—The shift toward data-intensive and Artificial Intelligence (AI)-driven cloud workloads has rendered traditional Central Processing Unit (CPU)-centric power modeling increasingly insufficient for accurate datacenter energy estimation. While CloudSim is the de facto standard for evaluating energy-aware strategies, its native models predominantly focus on CPU utilization, effectively overlooking the non-trivial power contributions of auxiliary subsystems. This paper presents a modular multi-resource power modeling framework for CloudSim that integrates subsystem-level energy estimation into a unified simulation workflow. The framework introduces three critical extensions: (i) a high-resolution monitoring module capturing time-series utilization for CPU, Random Access Memory (RAM), bandwidth, and storage; (ii) modular, resource-specific power models that facilitate disaggregated energy attribution; and (iii) enhanced datacenter components supporting multi-dimensional resource capacities and tiered energy analytics.By providing a cohesive abstraction layer and unified Application Programming Interface (API), the framework enables consistent and extensible energy evaluation across heterogeneous infrastructure scales. Evaluation under high-intensity workloads indicates that traditional CPU-centric models obscure a non-trivial share of subsystem-level energy consumption, with non-CPU components representing approximately one-third of the total energy footprint in the evaluated stress-test scenario.**

Keywords-Cloud computing; energy efficiency; power modeling; multi-resource simulation; CloudSim; datacenter sustainability.

I. INTRODUCTION

The digital transformation of the global economy, accelerated by Artificial Intelligence (AI)-driven workloads and data-intensive services, has necessitated a critical re-evaluation of datacenter energy efficiency. Recent projections estimate that datacenters accounted for approximately 1.5% of global electricity consumption in 2024, with a trajectory nearing 3% by 2030 [1]. Within these infrastructures, Information Technology (IT) equipment represents the primary energy sink [2]. While the Central Processing Unit (CPU) has historically been viewed as the dominant contributor, empirical studies increasingly show that memory, storage, and network subsystems account for a substantial and growing share of total system energy [3][4].

Despite this shift, current simulation-based research remains largely tethered to *CPU-centric power models*. These models estimate host power solely as a function of CPU utilization, implicitly assuming that auxiliary subsystems are either negligible or scale proportionally with computational load. Measurement-based studies contradict this assumption, demonstrating that memory refresh operations, storage I/O, and network traffic exhibit power behaviors that are weakly correlated or entirely decoupled from CPU activity [5][6]. Consequently, relying on CPU-only models introduces a visibility gap that can lead to skewed energy evaluations and misleading conclusions when testing energy-aware cloud management policies [7].

CloudSim [8] is the most widely used simulation toolkit for cloud resource management research. However, its native power modeling capabilities focus primarily on CPU utilization and do not provide an integrated mechanism for aggregating subsystem-level energy consumption. Although several extensions exist that model individual resources such as networking or storage [9][10], these solutions are typically domain-specific and operate in isolation. As a result, researchers lack a unified and extensible framework for holistic host-level energy estimation within CloudSim.

To address this limitation, this paper proposes a modular multi-resource power modeling framework for CloudSim. The contribution of this work lies in the integration and unification of subsystem-level power models into a cohesive simulation framework, building upon existing empirical modeling approaches [5][6]. The framework enables fine-grained energy attribution across CPU, memory, storage, and network subsystems while preserving compatibility with existing CloudSim scheduling and allocation policies.

The key contributions of this work are:

- *A unified multi-resource simulation framework* that integrates CPU, memory, storage, and network power estimation within CloudSim.
- *Granular energy attribution* at the virtual machine (VM), host, and datacenter levels to improve interpretability and reproducibility of simulation studies.
- *Quantification of modeling bias*, demonstrating that CPU-only models structurally misestimate energy consumption under data-intensive workloads.

The remainder of this paper is organized as follows. Section II reviews related work on datacenter power modeling and CloudSim extensions. Section III introduces the CPU-centric baseline models used for comparison. Section IV presents the proposed multi-resource framework and its integration into CloudSim. Section V describes the experimental setup and workload design. Section VI presents and discusses the evaluation results. Finally, Section VII concludes the paper and outlines future research directions.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

II. Background and Related Work

This section reviews prior work on server power modeling and cloud simulation, highlighting the need for a unified multi-resource energy model.

A. Power Consumption Dynamics in Modern Servers

Modern cloud servers are composed of several power-consuming subsystems, including Central Processing Units (CPUs), memory, storage, and network interfaces. While CPUs have historically been the primary focus of energy modeling, recent empirical studies demonstrate that non-CPU components now contribute a substantial and growing portion of total energy consumption by the servers [3][4]. Memory subsystems consume significant power due to constant background refresh cycles, even during low utilization; storage devices incur energy costs during state transitions and Input/Output (I/O) operations; and network interfaces draw power relative to data transfer rates [5][6]. Crucially, these components exhibit power behaviors that are weakly correlated with CPU utilization, rendering traditional CPU-only models structurally incapable of reflecting true server power dynamics.

B. Cloud Simulation Frameworks and Extensions

CloudSim [8] is the most widely adopted toolkit for evaluating cloud resource management strategies, including allocation, scheduling, and energy-efficient consolidation. Its modular design has enabled several domain-specific extensions, such as CloudSim Plus [11] for structural improvements, NetworkCloudSim [9][12] for communication modeling, and CloudSimDisk [10] for disk I/O simulation. While these tools improve modeling fidelity within individual subsystems, they typically operate in isolation and do not provide an integrated host-level energy aggregation layer.

Other simulation frameworks, such as GreenCloud and iFogSim, focus on network-centric and edge computing scenarios, respectively. However, none of these frameworks offer a unified and extensible abstraction layer that aggregates CPU, memory, storage, and network power into a single host-level energy profile. As a result, most CloudSim-based energy-aware studies continue to rely on CPU-centric power models despite well-documented subsystem-level power diversity.

C. Single-Resource vs. Multi-Resource Energy Modeling

Energy-aware cloud research has traditionally emphasized CPU-oriented techniques, such as Dynamic Voltage and Frequency Scaling (DVFS), utilization-based scheduling, and virtual machine consolidation [13]. Evaluating these techniques through CPU-only power models implicitly assumes that non-CPU energy scales with computational load. Measurement-based studies contradict this assumption, demonstrating independent and often non-linear power behaviors across CPU, memory, and network subsystems [5][6].

Recent simulation efforts have begun to incorporate selected non-CPU components [14]. However, these approaches typically lack full subsystem coverage, unified aggregation, or modular extensibility, limiting their applicability for evaluating cross-resource energy-management strategies.

D. Research Gap and Motivation

Despite growing recognition of multi-resource energy behavior, existing simulation approaches lack a comprehensive and modular solution for host-level power modeling in CloudSim. Specifically, there is an absence of: (i) an integrated model encompassing CPU, memory, storage, and network power, (ii) a unified host-level energy aggregation layer, and (iii) fine-grained subsystem-level attribution. These limitations motivate the need for a simulation framework that coherently integrates existing subsystem models into a unified, extensible energy estimation workflow.

III. CPU-Centric Baseline Models

To establish a reference for comparison, we employ the two most commonly used CPU-centric power models in CloudSim-based energy-aware studies. These models are not incorrect per se, but they abstract host power consumption as a function of CPU utilization alone, implicitly aggregating all subsystem energy into a single dimension. They therefore serve as appropriate baselines for quantifying the modeling bias introduced by single-resource assumptions.

A. Linear CPU-Only Power Model

The linear model estimates host power by interpolating between idle and maximum CPU load, treating all non-CPU power as a fixed component of the idle state. This approach is widely adopted due to its simplicity and low computational overhead.

$$P_{\text{host}}^{\text{lin}}(t) = P_{\text{idle}} + \left(P_{100} - P_{\text{idle}}\right) U_{\text{cpu}}(t) \tag{1}$$

where P_{idle} and P_{100} represent the measured host power at 0% and 100% CPU utilization, respectively, and $U_{\text{cpu}}(t)$ denotes the instantaneous CPU utilization. By construction, this model assumes that memory, storage, and network subsystems scale proportionally with CPU load, an assumption that does not hold under data-intensive workloads.

B. SPECpower-Based CPU Model

The SPECpower-based model relies on empirical measurements obtained from the SPECpower_ssj2008 benchmark [15]. It maps discrete CPU utilization levels to measured power values and interpolates between them to obtain a continuous power curve.

$$P_{\text{host}}^{\text{spec}}(t) = \text{SPEC}\big(U_{\text{cpu}}(t)\big) \tag{2}$$

where $\text{SPEC}(\cdot)$ denotes the benchmark-derived lookup and interpolation function. Although this model captures non-linear CPU power behavior more accurately than the linear approach, it still encapsulates the entire host energy consumption within a single CPU-driven function.

Both baseline models are computationally efficient and widely accepted in the literature. However, neither provides visibility into the independent contributions of non-CPU subsystems, nor can they capture energy variations caused by memory-intensive, I/O-intensive, or network-heavy workloads.These limitations motivate the need for a multi-resource modeling approach.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

IV. Multi-Resource Power Modeling Framework

To address the limitations of CPU-only host power abstraction, we extend CloudSim with a unified Multi-Resource (MR) power modeling framework that aggregates subsystem-level energy estimation into a single host-level profile. The objective of the framework is not to redefine physical power models, but to coherently integrate established subsystem-level models into a modular and extensible CloudSim-compatible architecture.

A. Framework Overview

The proposed framework extends CloudSim with a modular, multi-resource energy modeling layer designed to: (i) provide holistic host power estimation through subsystem-specific models, (ii) support heterogeneous hardware configurations, (iii) enable modular substitution or calibration of subsystem models, and (iv) support VM, host, and datacenter-level energy accounting.

As shown in Figure 1, the framework is implemented as a non-intrusive extension layer that interfaces with native CloudSim entities through well-defined APIs. A Monitoring Module periodically retrieves fine-grained utilization metrics from hosts and VMs, including Central Processing Unit (CPU) utilization, Random Access Memory (RAM) usage, storage I/O activity, and Network Interface Card (NIC) traffic. These metrics are processed by independent subsystem power models, each responsible for estimating instantaneous power consumption for its corresponding hardware component.

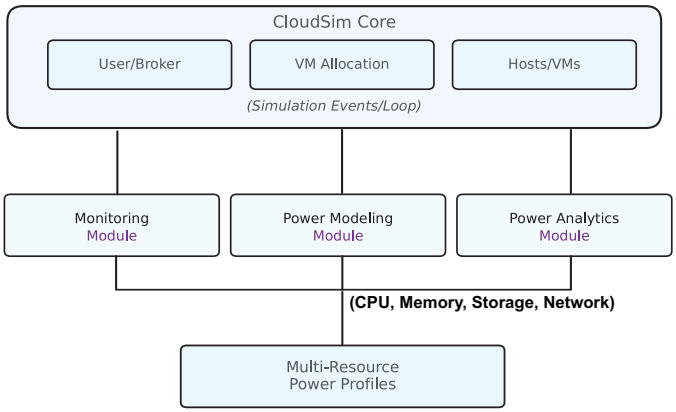


Figure 1. Architecture of the proposed multi-resource power modeling framework integrated into CloudSim.

The resulting subsystem-level power estimates are combined into a unified multi-resource power profile, which represents the dynamic host power behavior across all major subsystems. Interfaces are loosely coupled, allowing individual power models to be replaced, recalibrated, or extended without modifying CloudSim's internal scheduling or allocation logic.

B. Power Modeling Module

At each simulation timestep t, the total host power is computed as the aggregation of subsystem-level contributions:

$$P_{\text{host}}(t) = P_{\text{CPU}}(t) + P_{\text{RAM}}(t) + P_{\text{HDD}}(t) + P_{\text{NET}}(t) \quad (3)$$

Equation (3) reflects a structural decomposition of host power into major hardware subsystems. The formulation does not assume strict physical independence; rather, it provides a modular abstraction that avoids implicit bundling of non-CPU power into a single utilization variable. Subsystem models are parameterized and can be calibrated to represent different hardware platforms.

CPU Subsystem: In the proposed framework, the CPU model represents only the computational core's footprint. Non-CPU idle power is excluded to prevent double-counting across subsystems.

$$P^{\text{lin}}_{\text{CPU}}(t) = P_{\text{CPU,idle}} + \big(P_{\text{CPU,max}} - P_{\text{CPU,idle}}\big)U_{\text{cpu}}(t) \quad (4)$$

$$P^{\text{spec}}_{\text{CPU}}(t) = \text{SPEC}_{\text{CPU}}\big(U_{\text{cpu}}(t)\big) \quad (5)$$

The CPU parameters are calibrated to represent CPU-only consumption rather than full-host measurements, ensuring consistency with 3.

Memory (RAM) Subsystem: Memory power is modeled as the sum of static refresh power and dynamic access-related power [16]:

$$P_{\text{RAM}}(t) = P_{\text{refresh}} \cdot M_{\text{alloc}} + P_{\text{active/GB}} \cdot M_{\text{active}}(t) \quad (6)$$

where M_{alloc} denotes allocated memory capacity and $M_{\text{active}}(t)$ denotes actively accessed memory at time t. The coefficients were selected based on values reported in prior measurement-based studies and scaled proportionally to the simulated hardware specifications.

Storage (HDD) Subsystem: Disk power is represented using a state-based abstraction:

$$P_{\text{HDD}}(t) = \alpha(t)P_{\text{active}} + \beta(t)P_{\text{idle}} + \gamma(t)P_{\text{startup}} \quad (7)$$

where $\alpha(t)$, $\beta(t)$, and $\gamma(t)$ denote the proportion of time spent in active, idle, and startup states during the sampling interval.

State proportions are estimated from simulated I/O throughput and request patterns. Alternative storage technologies (e.g., SSDs) can be incorporated by replacing the state parameters and power coefficients.

Network (NIC) Subsystem: NIC power is estimated using a throughput–efficiency relationship [6]:

$$P_{\text{NET}}(t) = P_{\text{nic,idle}} + \frac{R(t)}{\eta(R)} \quad (8)$$

where $R(t)$ represents network throughput and $\eta(R)$ denotes throughput-dependent energy efficiency. This formulation captures dynamic power scaling with traffic intensity while maintaining a configurable idle baseline. The efficiency function can be parameterized to reflect different link speeds and hardware generations.

C. Power Analytics Module

The Power Analytics Module aggregates subsystem-level power estimates to compute cumulative energy consumption at the VM, host, and datacenter levels. It supports detailed energy breakdowns by resource type, comparative

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

evaluation against baseline models, and workload-level energy profiling. This enables quantitative analysis of energy-aware scheduling, consolidation, and optimization techniques under heterogeneous workload conditions.

D. Integration with CloudSim

The framework is integrated into CloudSim as an extension layer without altering its core event handling, scheduling, or VM allocation mechanisms. Custom host and datacenter classes override native power accounting methods and redirect energy queries to the multi-resource aggregation function defined in (3), without modifying CloudSim's core scheduling logic.

The Monitoring Module operates at fixed simulation intervals and retrieves utilization metrics directly from CloudSim entities, ensuring compatibility with existing VM placement and consolidation policies. This design preserves backward compatibility while enabling enhanced visibility into non-CPU energy consumption.

V. Experimental Methodology

This section details the simulation environment, datacenter configurations, and the stress-test workload designed to evaluate the behavioral differences between the proposed multi-resource framework and traditional CPU-centric baselines. The objective is comparative analysis of modeling structure rather than validation against a specific physical deployment.

A. Simulation Environment and Datacenter Setup

Experiments were conducted using CloudSim extended with our multi-resource modeling layer. To assess the framework across varying infrastructure scales, we simulated three heterogeneous datacenters (DC1–DC3), representing large, medium, and small environments, respectively. Each datacenter comprises an equal distribution of four server models, with hardware specifications detailed in Table I.

Subsystem-level power coefficients for CPU, RAM, storage, and network components were selected from measurement-based studies reported in the literature and scaled proportionally to the hardware capacities shown in Table I. The same coefficient configuration was consistently applied across all datacenters to ensure controlled comparison between modeling approaches.

TABLE I. Hardware specifications of the four server models used in all datacenters (HP ProLiant ML110 G3/G4/G5 and a custom Celsius V840-based model).

Model	Cores	MIPS	RAM (MB)	Storage (GB)	NIC (Mbps)
ML110 G3	2	6000	4000	160	1000
ML110 G4	2	3720	4000	160	1000
ML110 G5	2	5320	4000	146	1000
Celsius V840	4	10400	16000	80	1000

B. Workload Characteristics and Experimental Design

To evaluate the framework's sensitivity to multi-dimensional resource demand, we utilize a high-intensity workload profile. Unlike standard CloudSim traces that primarily emphasize CPU-bound tasks, this experiment features a dynamic arrival pattern of 2,000 VMs configured with a 200% increase in baseline bandwidth and I/O requirements. This stress-test scenario is intended to force significant utilization of the RAM, Network, and Storage subsystems, thereby exposing structural limitations of models that ignore these components.

The workload is synthetically constructed to isolate subsystem-dominant behavior under controlled conditions. While real-world traces may exhibit more complex interdependencies, this design enables clearer observation of modeling bias under resource-intensive scenarios.

In this study, we maintain all hosts in an active state and explicitly disable power-saving optimizations (e.g., VM consolidation or DVFS). This experimental design allows us to quantify the baseline energy footprint of each modeling approach independent of scheduling effects, and to isolate differences arising purely from power estimation structure across the different datacenter scales summarized in Table II.

TABLE II. Summary of the three simulated datacenters with evenly distributed server types.

Parameter	DC1 (Large)	DC2 (Medium)	DC3 (Small)
Total Servers	1200	600	400
Servers per Type	300 each	150 each	100 each
CPU Cores	3000	1500	500
RAM (GB)	8,400	4,200	2,800
Total BW (Gbps)	1,200	600	400

Simulations were executed over a 24-hour duration with a 5-second monitoring interval. This high-resolution sampling is critical for capturing transient spikes in memory and network activity, which are often smoothed over or lost in traditional 1-minute or 5-minute sampling windows.

C. Energy Computation

For the proposed MR model, host power is computed according to 3. Cumulative energy consumption (E) is derived from the instantaneous power samples using trapezoidal numerical integration to ensure numerical stability over the 24-hour simulation period:

$$E = \sum_{t=1}^{T-1} \frac{P(t) + P(t+1)}{2} \cdot \Delta t, \tag{9}$$

where $P(t)$ represents the power at time t and Δt is the 5-second sampling interval.

D. Evaluation Metrics

To quantify the comparative behavior of the modeling approaches, we report three primary metrics:

- **Total Datacenter Energy (kWh):** The aggregate energy consumption estimated by the Linear, SPECpower, and MR models.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

- **Subsystem Energy Breakdown:** A granular percentage-based attribution of energy to CPU, RAM, Network, and Storage resources under the MR framework.
- **Relative Deviation:** The percentage deviation of CPU-only baselines relative to the MR reference model, calculated as:

$$\text{Deviation } (\%) = \frac{E_{\text{CPU-only}} - E_{\text{MR}}}{E_{\text{MR}}} \times 100. \quad (10)$$

The MR model is treated as a structurally richer reference for comparison rather than as empirical ground truth. Reported deviations therefore indicate differences in modeling abstraction rather than absolute physical measurement error.

VI. Results and Discussion

This section evaluates the behavior of the proposed MR model against traditional Linear and SPECpower baselines under the high-intensity stress-test conditions previously defined. Rather than treating the MR formulation as physical ground truth, the analysis uses it as a structurally disaggregated reference model to examine how CPU-centric approaches behave when non-CPU activity is significant.

A. Datacenter Energy: CPU-only vs. Multi-Resource

Table III summarizes the total energy consumption for the three simulated datacenters (DC1–DC3), enabling a direct comparison of the relative deviation exhibited by single-resource models. Figure 2 visualizes these differences across infrastructure scales.

TABLE III. Energy consumption and relative deviation of CPU-only models compared to the MR baseline.

DC	MR	CPU-only Linear		CPU-only SPEC	
	Energy (kWh)	Energy (kWh)	Error (%)	Energy (kWh)	Error (%)
DC1	881.4	903.4	+2.49	834.2	-5.35
DC2	721.4	739.5	+2.51	711.0	-1.44
DC3	716.5	734.4	+2.50	714.0	-0.35

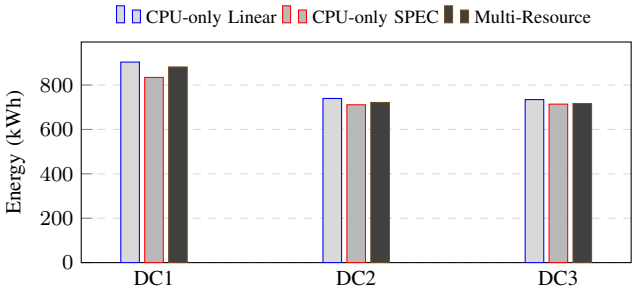


Figure 2. Comparative energy estimation across different models and datacenter scales.

Across all datacenter scales, the Linear baseline consistently shows a positive deviation of approximately +2.5%, while the SPECpower-only baseline yields a negative deviation reaching -5.35% in the most resource-intensive configuration (DC1). These results indicate a systematic directional bias in CPU-centric models under workloads with significant memory and I/O activity.

It is important to emphasize that these deviations are inherently workload-dependent. Under CPU-dominant scenarios with limited memory or I/O pressure, the gap between CPU-only and multi-resource formulations would be expected to narrow. The stress-test configuration used here intentionally amplifies cross-subsystem activity in order to expose structural modeling differences.

B. Subsystem Energy Attribution

The MR formulation enables granular subsystem-level attribution, as detailed in Table IV.

TABLE IV. Subsystem-level energy breakdown (kWh / % contribution) under the MR model.

Subsystem	DC1 (Large)	DC2 (Medium)	DC3 (Small)
CPU	592.1 / 67.18	483.4 / 67.01	480.3 / 67.03
Memory (RAM)	238.0 / 27.00	195.1 / 27.04	193.6 / 27.02
Storage (I/O)	43.5 / 4.94	36.6 / 5.08	36.3 / 5.07
Network (BW)	7.8 / 0.88	6.3 / 0.87	6.3 / 0.88
Total	**881.4 / 100**	**721.4 / 100**	**716.5 / 100**

While the CPU remains the dominant contributor (approximately 67%), memory and storage together account for roughly one-third of total energy consumption in this scenario. Because these subsystems do not scale strictly proportionally with CPU utilization, their independent dynamics explain the structural deviation observed in CPU-only host curves. The results therefore highlight modeling sensitivity to subsystem disaggregation rather than establishing absolute physical correctness.

C. Model Comparison and Directional Bias

Our evaluation reveals a distinct directional bias in traditional single-resource models. The Linear model serves as a pessimistic baseline, consistently overestimating total energy by approximately 2.5%. This stems from its simplified allocation logic, which treats idle and dynamic non-CPU power as a proportional extension of CPU load. Conversely, the SPECpower-only baseline provides a significant underestimation (reaching -5.35% in DC1). By modeling host power solely through CPU-centric benchmarks, it overlooks the dynamic fluctuations of RAM and network activity, which in our high-intensity scenario represent approximately one-third of the energy footprint.

D. Modeling Assumptions and Limitations

The proposed framework enhances modeling realism through multi-resource power functions and subsystem-level energy accounting, but it abstracts OS scheduling, hardware-level effects, and relies on integrated power models. Consequently, results should be interpreted comparatively rather than as precise real-world predictions. The enterprise workload traces used provide realistic correlated resource behavior, though they do not represent emerging paradigms such as GPU-accelerated, tightly coupled HPC, or serverless workloads. Resource utilization is measured only during active

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

execution, improving interpretability while excluding long-term idle effects, and the model accounts solely for IT power, omitting thermal and facility overheads. Furthermore, subsystem power coefficients are derived from literature-based measurements and scaled proportionally to simulated hardware configurations rather than calibrated against direct telemetry from a specific physical deployment. As such, the MR model serves as a structurally richer analytical reference rather than an empirically validated ground-truth baseline. Nonetheless, the controlled and deterministic experimental design ensures valid, reproducible, and meaningful relative comparisons across configurations.

E. Key Takeaways

- **Subsystem visibility:** Multi-resource modeling exposes non-CPU energy contributions (e.g., 289.3 kWh in DC1) that are structurally hidden in CPU-only formulations.
- **Directional modeling bias:** CPU-centric baselines exhibit consistent over- or under-estimation tendencies under I/O-intensive conditions.
- **Scale consistency:** Relative deviation remains largely scale-independent across DC1–DC3, indicating that the observed behavior is driven by modeling structure rather than datacenter size.
- **Workload sensitivity:** The magnitude of deviation is scenario-dependent and is expected to diminish under CPU-dominant workloads with limited memory and I/O activity.

VII. Conclusion and Future Work

This paper presented a modular multi-resource power modeling framework for CloudSim that extends traditional CPU-centric energy simulation. By disaggregating host power into CPU, memory, storage, and networking components, the framework enables structured energy accounting and subsystem-level attribution within simulation studies.

Experimental results under high-intensity data-driven workloads indicate that traditional Linear and SPECpower-based models may deviate by up to approximately 5% when compared to the proposed multi-resource formulation. This deviation arises primarily from the exclusion of memory and I/O dynamics, which accounted for roughly one-third of total energy consumption in the evaluated scenario. These findings highlight the sensitivity of energy estimation to modeling structure rather than asserting absolute physical accuracy.

The framework is intended as a modular research instrument for subsystem-aware policy exploration. Future work may include hardware-level validation using empirical power measurements and telemetry traces to calibrate subsystem coefficients and strengthen realism, as well as support for heterogeneous accelerators and joint energy–performance optimization strategies that consider cross-subsystem interactions.

References

[1] I. E. A. (IEA), "Energy and AI – Analysis," International Energy Agency, 2025, [Retrieved: 02, 2026]. [Online]. Available: https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai

[2] C. Jin, X. Bai, C. Yang, W. Mao, and X. Xu, "A review of power consumption models of servers in data centers," *Applied Energy*, vol. 265, p. 114 806, 2020, ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2020.114806

[3] L. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007. DOI: 10.1109/MC.2007.443

[4] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "Chapter 3 - A taxonomy and survey of energy-efficient data centers and cloud computing systems," in *Advances in Computers*, M. V. Zelkowitz, Ed., vol. 82, Elsevier, 2011, pp. 47–111. DOI: 10.1016/B978-0-12-385512-1.00003-7

[5] R. Basmadjian, N. Ali, F. Niedermeier, H. De Meer, and G. Giuliani, "A methodology to predict the power consumption of servers in data centres," in *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, New York USA: ACM, 2011, pp. 1–10, ISBN: 978-1-4503-1313-1. DOI: 10.1145/2318716.2318718

[6] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A measurement-based characterization of the energy consumption in data center servers," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2863–2877, 2015, ISSN: 1558-0008. DOI: 10.1109/JSAC.2015.2481198

[7] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016, ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2481183

[8] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software - Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. DOI: 10.1002/spe.995

[9] R. Malhotra and P. Jain, "Study and comparison of cloudsim simulators in the cloud computing," *The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)*, vol. 1, no. 4, pp. 111–115, 2013. DOI: 10.9756/SIJCSEA/V1I4/0104510201

[10] B. Louis, K. Mitra, S. Saguna, and C. Åhlund, "CloudSimDisk: Energy-aware storage simulation in cloudsim," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 11–15. DOI: 10.1109/UCC.2015.15

[11] M. C. Da Silva Filho, "CloudSim Plus: A modern, extensible simulation framework for cloud computing and container-based infrastructures," *IEEE Latin America Transactions*, vol. 15, no. 3, pp. 451–458, 2017. DOI: 10.1109/TLA.2017.7867605

[12] M. H. Habaebi et al., "Extending CloudSim to simulate sensor networks," *SIMULATION*, vol. 99, no. 1, pp. 3–22, 2023. DOI: 10.1177/00375497221105530

[13] R. Muralidhar, R. Borovica-Gajic, and R. Buyya, "Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards," *ACM Computing Surveys*, vol. 54, no. 11, pp. 1–37, 2022, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3511094

[14] K. Farah, "Towards energy efficient cloud data centers - A framework for evaluation and analysis of energy efficiency," [Retrieved: 02, 2026], Master thesis, Oslo Metropolitan University (OsloMet), 2024.

[15] SPEC, *SPECpower_ssj 2008*, [Retrieved: 02, 2026]. [Online]. Available: https://www.spec.org/power_ssj2008/

[16] M. Moreau, "Estimating the energy consumption of emerging random access memory technologies," [Retrieved: 02, 2026], Master thesis, Institutt for elektronikk og telekommunikasjon, 2013. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2370830

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Accelerated Flow Processing in Kubernetes Overlay Networks

Srinath Vasudevan, Khaled Harfoush
Department of Computer Science
North Carolina State University
Raleigh, USA
e-mail: svasude5@alumni.ncsu.edu, kaharfou@ncsu.edu

***Abstract*—Kubernetes is a popular container orchestration tool which makes managing numerous containers simple through various abstractions. *Pods* are the abstraction for containers, and Kubernetes allows for pod inter-networking through the Container Network Interface (CNI) specification. Overlay networking is a technique which utilizes network tunneling across hosts to allow pods to communicate with each other using their private IP addresses. Previous work shows that overlay networking results in significant network performance degradation due to the packet encapsulation and decapsulation processing steps required for network tunneling, largely because all packet processing is usually performed on a single CPU core by default. Optimizations in the Linux kernel exist, such as Receive Packet Steering (RPS) and Receive Flow Steering (RFS), in order to parallelize packet processing per flow. The current literature does not thoroughly evaluate RFS, nor does it evaluate RPS when enabled at the container level in overlay networks. In this paper, we conduct a thorough measurement study and a performance analysis of these optimizations. Our results demonstrate the greatest improvements in average bitrate and CPU core load distribution when RFS is enabled at the host and container levels, although improvements are seen in other scenarios as well. This important work outlines actionable methods for overlay network performance gains that are applicable to public cloud systems.**

Keywords-Kubernetes; Calico; Overlay Networks; Receive Flow Steering; Receive Packet Steering.

I. INTRODUCTION

Virtual *containers* have seen increased popularity in recent years. Unlike Virtual Machines (VMs), containers virtualize the host kernel rather than the hardware. This allows containers to be smaller and have a faster creation/termination time than their VM counterparts, leading to increased portability and flexibility. Multiple technologies have been developed around containers, namely container orchestration tools, such as Kubernetes [1] and OpenShift [2], allowing for the simplified management of multiple containers in a centralized environment. A standardized specification for container networking called the Container Network Interface (CNI) [3] was created as a project by the Cloud Native Computing Foundation (CNCF) [4] in an attempt to have a standardized container networking interface. Furthermore, a wide variety of solutions were introduced to enable inter-container communication such as Calico [5], Cilium [6], and Flannel [7]. These efforts have led to the wide adoption of container orchestration tools throughout production workloads [8].

However, container orchestration tools introduce some complexity in scheduling, service routing, and communication [9], [10]. A common implementation of container networking uses a strategy called *overlay networking*, which relies on network *tunneling*, allowing the use of private IP addresses to communicate over a public network. Network tunneling encapsulates/decapsulates packets to preserve private IP addresses when transmitted over a public network. Overlay networks incur a longer data path due to this encapsulation/decapsulation necessity [9]. The elongated data path is the main contributor to the increased latencies seen in overlay networks [10]. Another problem with the longer data path is that packets are normally only processed serially on a single core, leading to a bottleneck in network throughput.

Certain technologies exist to mitigate the impact of serial packet processing. Receive Side Scaling (RSS) [11] is a hardware optimization that uses multiple receiving queues on a single NIC to accelerate packet processing. Receive Packet Steering (RPS) [12] is a software implementation of RSS that aims to distribute packet processing across many cores. Receive Flow Steering (RFS) [13] is a software extension to RPS that tries to improve RPS by processing packets on the core of the destination application buffer in order to benefit from cache locality. Both software options are supported by the Linux kernel [14] and need to be explicitly enabled on applicable network devices.

However, current research efforts do not evaluate the efficacy of RFS in a container overlay network, nor do they evaluate the efficacy of RPS and RFS when enabled at the container level [15]. Additionally, current efforts are not clear on the configurations used for RPS [15]. Overall, the benefits of these existing optimizations are inadequately studied. In this paper, we fill this gap by evaluating the benefits of introducing the RPS and the RFS optimizations in various configurations on a VXLAN-based Calico overlay network [5] using Kubernetes for container orchestration. Our work provides insights into scenarios where RPS and RFS perform the best and the inherent incompatibilities of these optimizations in overlay networks, allowing future researchers to obtain a clear idea of the current state of overlay network optimizations.

The remainder of this paper is organized as follows. In Section II, we survey related work. In Section III, we introduce basic terminology regarding Kubernetes, network tunneling, and overlay networking. In Section IV, we provide necessary background about our chosen overlay network, RPS, and RFS. In Section V, we evaluate the performance of the RPS and RFS optimizations. We finally conclude in Section VI.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

II. Related Work

Many technologies exist to support communication between hosts including Network Address Translation (NAT), host networking, routing via BGP, and overlay networks [10]. Among these, overlay networks are very common especially in container orchestration environments due to their flexibility and straightforward deployment.

Studies were conducted to compare the performance of these technologies [10] [16]. In [10], Suo et al. analyzed various methods for intra and inter-host networking among containers under various protocols and network conditions. Among the overlay networks analyzed were Calico [5], Weave [17], Flannel [7], and Docker Overlay [18]. Compared to other networking options, overlay networks performed worse in terms of throughput, latency, and network launch time. However, comparing overlay network performance to technologies, such as NAT or host mode networking is an unfair comparison as they are built for different purposes. Overlay networking provides an easy-to-use networking setup among containers, which is easier to manage in changing network topologies. In [9], Suo et al. studied the reasons for the network overhead in overlay networks. They observed high amounts of scheduled hardware and software interrupts in overlay networks compared to regular host networking. This increase coupled with the fact that multicore systems are unable to effectively parallelize the excessive software interrupts without specialized hardware or protocols leads to an imbalance of compute power in such systems. This is largely attributed to the longer data path of a packet due to the added overhead from encapsulation/decapsulation and the container bridge network. In [19], Lei et al. highlighted that RPS and RFS are both flow-level parallelization methodologies, and suggested that packet-level parallelization would increase performance for single-flow situations. The authors also found that multiple container flows do not saturate a particular network link, indicating that overlay networks do not scale well, likely due to the large amount of context switches. This phenomenon is further exacerbated with small packet sizes.

Various efforts were aimed at optimizing overlay network performance, whether offering software solutions or hardware solutions. On the **software** solutions front, in [20], Lin et al. propose SlimFast, which uses a host machine's socket directly from the container to eliminate encapsulation overhead while preserving the ability to utilize a container's private IP address without knowledge of the underlying host IP. However, SlimFast intercepts system calls related to socket communication, which requires explicit kernel support. In [15], Lei et al. propose mFlow, a technology implementing packet-level parallelism [19] for *stateless* steps in the networking stack. These include steps where in-order processing is unnecessary. For steps that require in-order processing (such as TCP or L7 protocols), the packets are reassembled in order before that step is processed to preserve order. Instead of processing an entire flow in parallel to other flows, mFlow breaks up a single flow into subflows and processes the subflows in parallel. Naturally, the subflows retain their ordering. mFlow improves TCP and UDP throughput when compared to a regular overlay networks. Performance increases are also consistently seen when compared to an RPS enabled overlay network. Proposed solutions do not evaluate the efficacy of RFS and all include technologies that need explicit kernel support. RPS and RFS are already supported by the Linux kernel [14], so they can readily be used in public cloud environments - an effort which we undertake in this paper.

On the **hardware** solutions front, hardware-acceleration is an option. In [21], Ma et al. propose a SmartNIC-based software-hardware design to enhance the performance in a cloud environment. The main idea is to offload the encapsulation/decapsulation operations typically done by the kernel to the SmartNIC. While this proved to reduce CPU overhead while lowering latency and increasing throughput [21], the results show that throughput does not scale as well when the core count becomes high (greater than around 4-6 cores). This is attributed to the fact that SmartNIC does not distribute packets effectively to the different cores. Note that hardware offloading for container networks, as opposed to software solutions, leads to security concerns due to multi-tenancy. Finally, the adoption of SmartNIC hardware in a public cloud environment is challenging. Software solutions are readily supported by the Linux kernel, so any potential benefits can be readily adopted.

Overall, the issue of overlay network optimization has many solutions being researched. However, the existing optimizations available in the Linux kernel have not been exhaustively and adequately studied in this context. In this work, we systematically study RPS and RFS in various configurations to cover the gaps in previous work.

III. Terminology

In the following subsections, we introduce necessary terminology for understanding Kubernetes and overlay networking concepts.

A. Kubernetes

Kubernetes is a container orchestration framework that simplifies many management tasks of multi-container environments [1]. Kubernetes has many abstractions and various terms we use throughout this paper. These are defined as follows. A *Cluster* is a logically isolated Kubernetes environment, consisting of one or more host machines; a.k.a. *nodes*. Nodes are able to communicate with each other to match the desired state of the cluster. Nodes are typically implemented as Virtual Machines. A *Pod* is the smallest Kubernetes abstraction, typically used to represent a single container. A pod can also contain multiple containers, but this is mainly common for monitoring purposes. In this paper, Kubernetes pods are synonymous to a single container. A *Service* is a networking abstraction which enables a set of pods to be accessible by a static IP address. This static IP routes to a set of pods automatically, and a DNS name for the service is also created. In this paper, each service we create

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

will only route to a single pod. *CNI Plugins* [3] enable inter-pod networking in Kubernetes. These do not come installed by default on Kubernetes and must be explicitly installed. Many different implementations exist. In our experiments, we use Calico [5].

B. Network Tunneling

Network tunneling [22] is a technology that allows for a packet of some protocol to be transmitted over a network by encapsulating it in the header of another protocol. This is useful in the event the network does not support the encapsulated protocol. At the destination, the packet is decapsulated and the original packet is retrieved for processing. The destination should be able to process the original packet and its protocol. Popular tunneling protocols include VXLAN [23], GRE [24], and IPIP [25].

C. Container Overlay Networks

In an overlay network, a virtual Layer 2 network device (such as a Linux bridge) is created to handle communication between containers. The containers' virtual devices are connected via a virtual ethernet (`veth`) pair. Overlay networks take advantage of network tunneling to allow pods on separate host machines to communicate using their private IPs without any explicit knowledge of the destination host's IP. Another network interface is used to perform encapsulation and decapsulation of packets. The encapsulated packet header contains the IP of the destination host which the destination pod resides in. This destination host IP is typically referenced through a distributed KV store, such as `etcd` [26].

In networking contexts, two types of interrupts are invoked: software interrupt requests (`softirqs`) and hardware interrupt requests (`hardirqs`). `hardirqs` are always invoked by hardware events, such as a packet arriving at a NIC. This triggers a hardware interrupt handler to handle the packet and place it in the proper buffer. A `softirqs` is invoked to actually process the packet once it is available to the kernel. Most network processing occurs as a result of `softirqs`, and they take on the majority of packet processing workloads.

In a receive path, a packet first traverses the host network stack once the NIC copies the packet into a buffer and raises the first `softirq` [9]. After the software interrupt handler is invoked, processing continues up to Layer 3 and layer 4. This is where the packet is decapsulated and put in a queue for the virtual Layer 2 device. Another `softirq` is raised to handle this inner packet, which is emitted to the container through the veth pair from the virtual Layer 2 device. This in turn invokes yet another `softirq` to process the packet in the container's network stack.

IV. Background

In the following subsections, we give essential background on our chosen overlay network, overlay network processing and existing software-based network processing optimizations.

A. Calico

We use Calico [5] as our overlay network. Calico creates a new network interface (virtual Layer 2 device) in the host network namespace named `cali*` to communicate with a pod. Each pod has its own network namespace and is able to communicate with `cali*` via a virtual ethernet (veth) pair. The network interface for this pair on the pod is `eth0`. The backend installed by Calico depends on the type of encapsulation used. We use Calico configured with VXLAN [23] encapsulation, so another network interface in the host network namespace named `vxlan.calico` is created. This is where encapsulation and decapsulation according to the VXLAN tunneling protocol occurs. When installing Calico with Kubernetes, there is no need to set up a separate datastore (such as `etcd`). There is an option to directly interface with the Kubernetes API server to find pod-to-host IP mappings, which makes management easier. We use this option in our experiments. Figure 1 shows the general data path of a pod (pod 1 on host A) sending a packet to another pod on another host (pod 2 on host B), as well as all the network devices it traverses.

B. Packet Processing Overhead

A received packet needs to traverse the entire host networking stack as well as the entire container networking stack, incurring significant overhead. Encapsulation/Decapsulation and the invocation of corresponding `softirqs` are the main contributors to the overhead. Up to 3x more software interrupt requests (`softirqs`) compared to a native host network receive path were observed in [9].

The main reason behind the overhead from increased `softirqs` comes from how Linux schedules `softirqs` onto cores. Typically, hardware interrupt requests (`hardirqs`) are processed on a single chosen core [9]. Once a NIC receives a packet, a `hardirq` is raised and processed on that particular core. Any `softirqs` that result from the `hardirq` are typically processed on the same core the `hardirq` was processed on. This means that the first `softirq` (and every subsequent one) are processed on the same core. Furthermore, for a single queue NIC, all `hardirqs` and `softirqs` must be processed serially on that single core, mostly to avoid out-of-order packet processing. In a multicore system this can easily exhaust a single core's resources while leaving other cores idle.

C. Receive Packet Steering (RPS)

Receive Side Scaling (RSS) [11] is a hardware optimization that allows `softirq` processing to be distributed among multiple cores. RSS works by having multiple input queues (usually bound by the number of cores) for a single NIC, which means that the content of each queue can be processed by a different core. Receive Packet Steering (RPS) [12] is the software implementation of RSS. Rather than requiring multiple receive queues, RPS creates a hash based on relevant Layer 3 and Layer 4 information (IP and port) to choose an arbitrary CPU to process the packet on. This is an example of flow-level parallelism, where packets of the same flow get

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

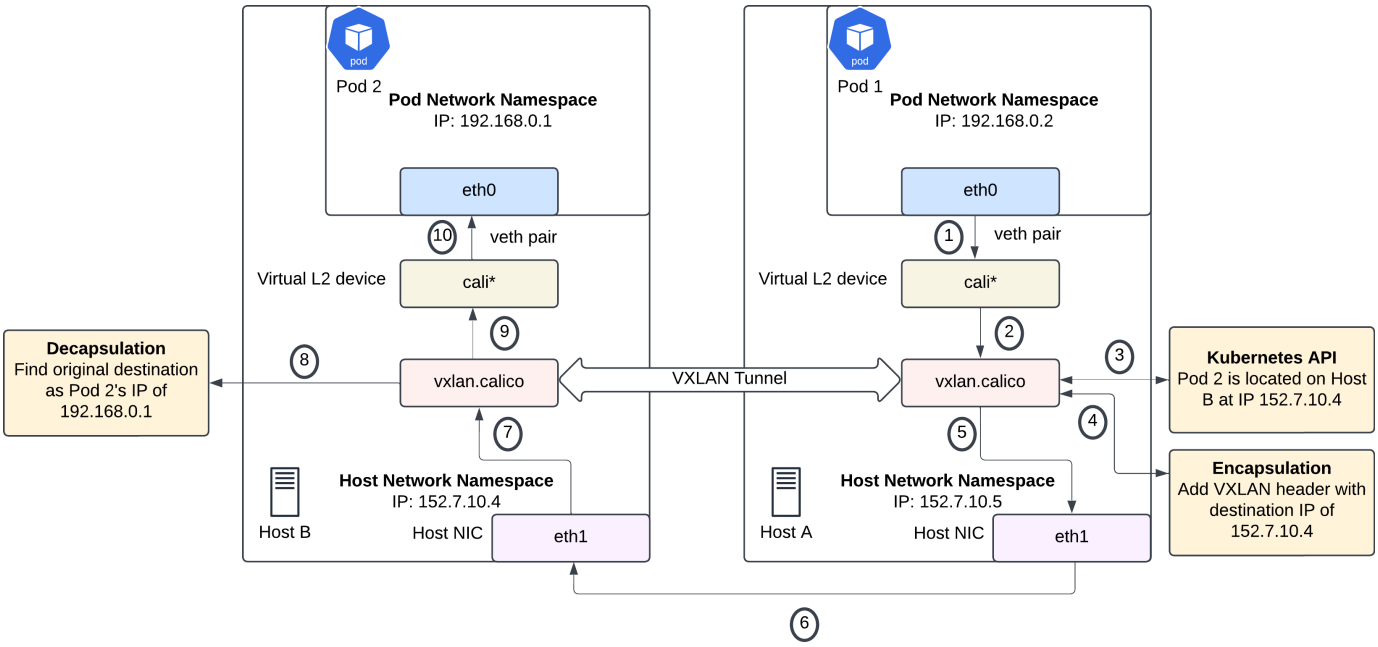


Figure 1. An example of how the Calico overlay network with VXLAN encapsulation transmits and receives packets. Pod 1 on Host A is sending a packet to Pod 2 on Host B.

processed on the same CPU. Figure 2 shows a simplified flow of a packet in a receive path with RPS enabled.

RPS must be explicitly enabled on a per-interface level. In order to do so, the file `/sys/class/net/{DEVICE}/queues/rx-{QUEUE-NUMBER}/rps_cpus` must be modified. In order to enable RPS on all cores of the eth0 device on its only receive queue, we write a value of `f` in the file `/sys/class/net/eth0/queues/rx-0/rps_cpus`.

D. Receive Flow Steering (RFS)

RFS [13] is an extension of RPS that aims to process a packet on the same CPU that the destination application runs on to take advantage of cache locality - although this is not guaranteed to happen. In order to achieve this goal, RFS uses one of the RPS generated hash tables and tracks the last used CPU for processing the flow. In order to enable RFS, we modify two files: `/proc/sys/net/core/rps_sock_flow_entries` and `/sys/class/net/{DEVICE}/queues/rx-{QUEUE-NUMBER}/rps_flow_cnt`. The former is typically set to 32768 to indicate the maximum number of connections. The latter indicates the number of expected flows per device queue. In a single queue device, this can be the same value as the `rps_sock_flow_entries`.

It should be noted that both RPS and RFS are processing techniques that exploit *flow parallelization*. Since the hash is based on Layer 3 and Layer 4 packet information, all processing for a single network flow is done on the same core. This is the easiest way to implement network processing parallelization as it avoids out-of-order packets. As a result of flow-level parallelization, RPS and RFS are not designed to increase performance in single-flow scenarios. Some techniques, such

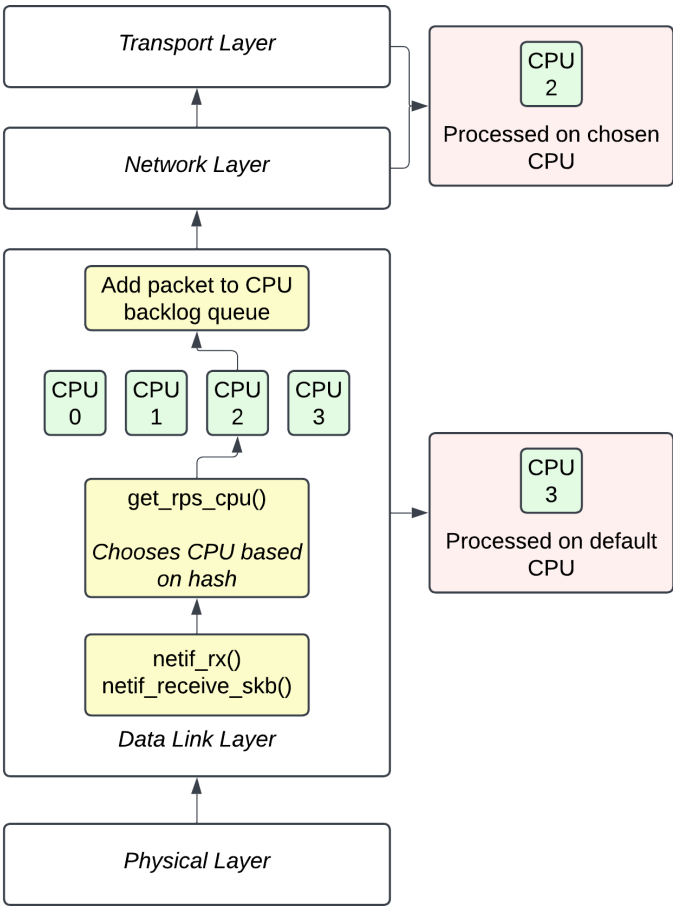


Figure 2. Simplified flow of a network receive path with RPS enabled on 4 cores in the Linux networking stack. The default core for packet processing is assumed to be CPU 3.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

as those proposed in [15] and [27], implement packet-level parallelism with methods to overcome out-of-order processing challenges.

V. Performance Evaluation

In the following subsections, we describe our experimental setup, evaluation tools, experiment details and the results.

A. Experimental Setup

The Kubernetes cluster we use consists of three nodes, where each node is a VM created with KVM as the hypervisor. One of these nodes is a control plane node, and the other two are exclusively used for the experiments (worker nodes) – Refer to Figure 3. Each VM is on a separate Dell PowerEdge R930 machine and is running Ubuntu 22.04 LTS. All VMs are provisioned with 4 cores and 8 GB of RAM. Each physical machine has a bonded network link to two top of rack switches. All machines we use are connected to the same two switches. We use `kubeadm` to install Kubernetes version 1.31.1 and use `containerd` as our container runtime. We installed the Calico CNI as our overlay network in a VXLAN encapsulation mode. All the code for the experiments can be found at this github repository [28].

In our experiments, we introduce different number of flows between the two worker nodes. A single flow between two worker nodes is a single client/server connection where the client and server continuously send packets to each other throughout the experiment duration. All clients are placed on one node and all servers are placed on the other node. This ensures that traffic flows from one VM to a separate VM. The number of clients and server pairs can be adjusted variably - we test number of pairs increasing by a power of 2 until we reach 16 client/server pairs. In this paper, we use "replicas" to mean the number of client/server pairs. For instance, 2 replicas means two clients and two servers in total.

B. Performance Metrics

For each experiment, we report on the following performance metrics: (1) The *normalized average bitrate*, (2) the *CPU Idle Percentage*, and (3) *CPU `softirqs` Percentage*. All our experiments are repeated to plot averages and 95% confidence intervals. The average bitrate in bits/sec for the duration of each experiment is normalized relative to the baseline experiment bitrate (when no RPS or RFS optimizations are deployed). A value larger than 1 reflects an improvement in bitrate over the baseline case. We rely on iPerf3, a network performance tool, to measure bitrates [29]. In our experiments, we set up an iPerf3 server on one pod and an iPerf3 client on another. From the client, we reference the server via its Kubernetes service's DNS name and let the client repeatedly send an array of 128 KB for 30 seconds over TCP. We then average the bitrate measured at each server instance. We use the mpstat [30] tool to collect CPU usage percentages by each core, and report on idle CPU percentages and the percentage of time a CPU is used to process `softirqs`. We run mpstat on the server node and use the `-P ALL` flag to report information for each core every second in each experiment, then compute averages.

C. Experiments

The overlay network configurations that we evaluate are: (1) **Baseline** (no RPS or RFS optimizations), (2) **RPS** (RPS enabled on all cores), (3) **RPS+** (RPS enabled on all cores and container-level RPS enabled), (4) **RFS** (RFS enabled), and (5) **RFS+** (RFS enabled and container-level RFS enabled). We do not optimize the baseline scenario in any unique way compared to the other scenarios.

We only enable RFS or RPS on the iPerf3 server's node exclusively on the following three network interfaces: (1) `vxlan.calico` interface: The VXLAN backend, (2) `cali*` interface: The Layer 2 switch to communicate with pods, and (3) `eth1` interface: The virtual NIC of the host machine. We keep these optimizations disabled on all other interfaces on the server node. For the RPS+ and RFS+ experiments, we enable the respective optimization at the container level in addition to the host interfaces. For example, RFS+ enables RFS at the three previously mentioned network interfaces as well as all interfaces in the container. This requires us to remount the `/sys` filesystem as read-write (containers have the `/sys` filesystem as read-only by default). In Kubernetes, the pod is given root and privileged access to perform this operation.

For the options with RPS enabled on all cores, a value of `f` is written to the `rps_cpus` file for all network interfaces for which we wish to enable RPS on. For options that enable RFS, the `rps_sock_flow_entries` file and `rps_flow_cnt` files for each network device are set to 32768 as described in Section IV.

D. Results

Figures 4 (a) and (b) plot the CPU idle time percentages for all cores when using 1 replica and 16 replicas, respectively. The figure reveals that a single replica does not stress the CPU and leads to more than 90% CPU idle time for all cores, while 16 replicas stress the cores leading to around 60% of idle time for cores 0, 1 and 2, and leading to only about 30% of idle time for core 3. As a result, single replica scenarios are not expected to benefit significantly from flow parallelization optimizations. This is largely due to the fact RPS and RFS perform much better with multiple flows since they rely on flow-level parallelization. Note that packet processing is done on Core 3 by default, evidenced by the very low idle core percentage for the baseline case. This can be further evidenced by inspecting Figures 5 (a) and (b), which plot the CPU percentage of time spent handling software interrupts (`softirqs`) for all cores when using 1 replica and 16 replicas, respectively. Focusing on the 16 replicas case, core 3 is handling more (`softirqs`) than the other cores. Also, note in Figure 5 (b), that all optimizations reduce the percentage of time that core 3 spends servicing `softirqs` compared to the baseline case. Among these optimizations, RFS+ offers the most reduction, and thus the best load balancing among the cores, followed by RFS and RPS+.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

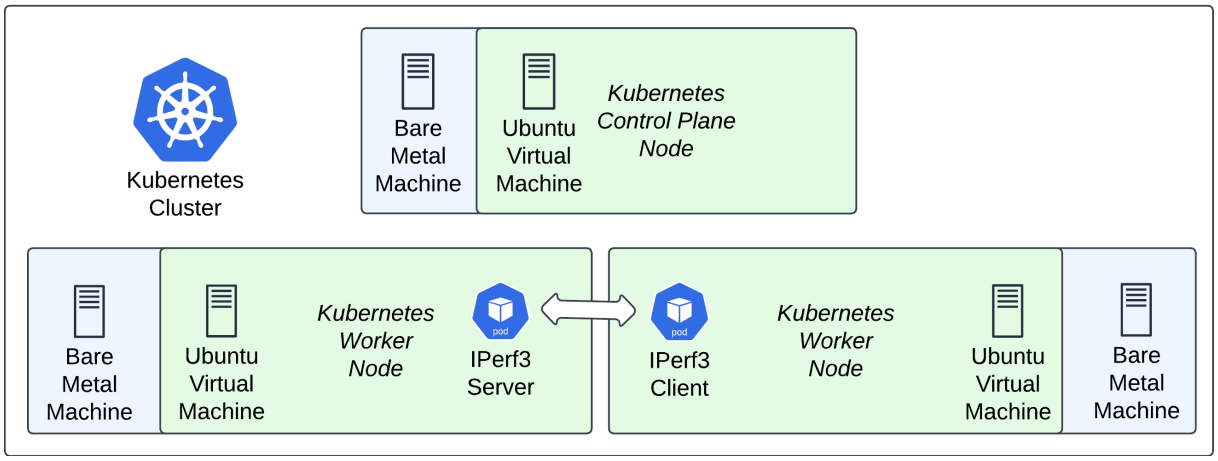


Figure 3. The general experiment setup in the Kubernetes environment.

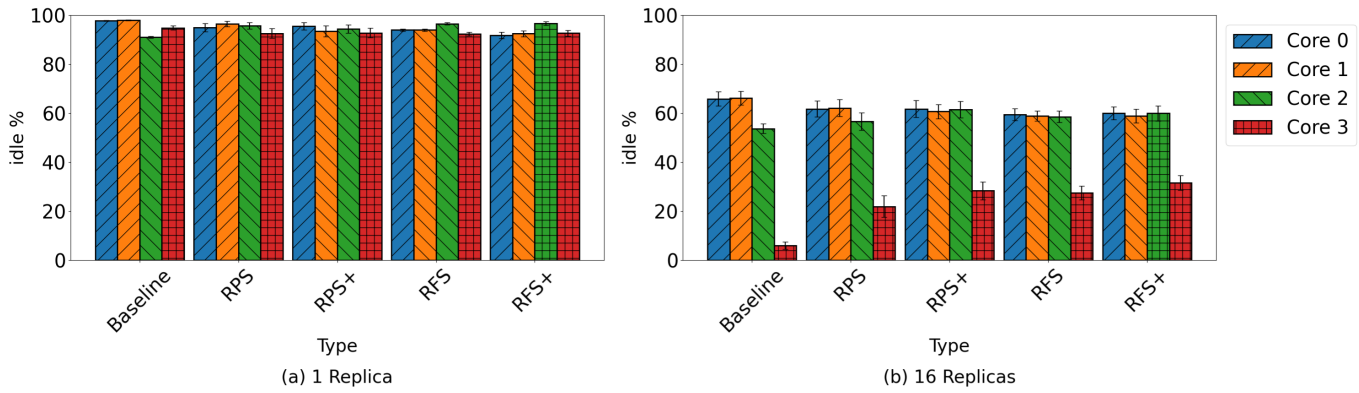


Figure 4. Average idle CPU percentage by cores for each optimization with (a) a single replica and (b) 16 replicas.

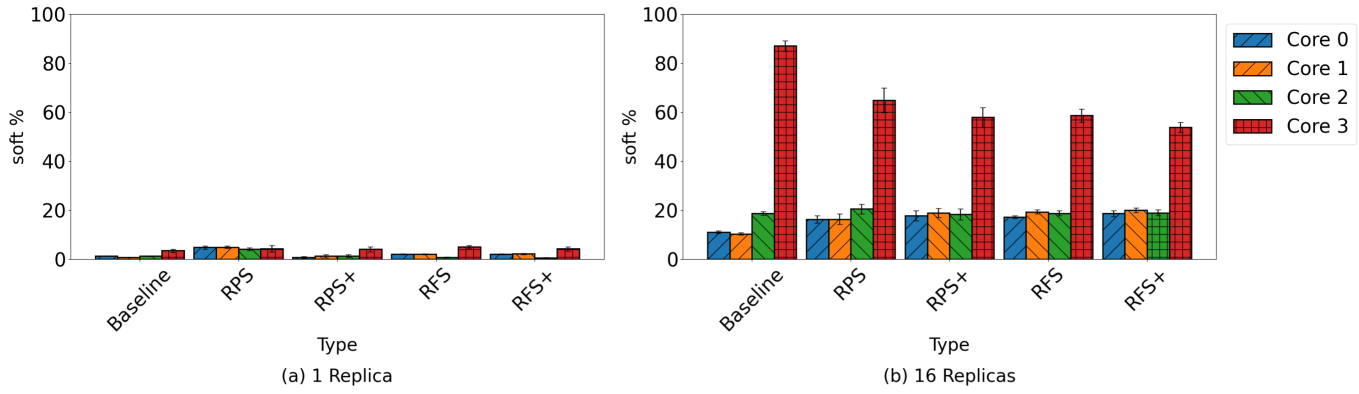


Figure 5. Average CPU percentage used for software interrupts by cores for each optimization with (a) a single replica and (b) 16 replicas.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

The reason we do not see even more balanced numbers in terms of core utilization is due to how RPS (and in turn, RFS) protocols work. The generated hash that determines the core to process a packet on relies on Layer 3/Layer 4 information, meaning that only the protocol processing steps (Layer 3/Layer 4) can be parallelized [12]. In an overlay network, the final packet processing step at the destination pod and the VXLAN decapsulation processing step are the only two steps that will process packets at the protocol level. All processing up to Layer 2, including the transmission of packets from the Layer 2 bridge to actual pods, is not steered to any other cores. The added step to transmit packets from the virtual Layer 2 bridge to the pods leads overlay networks to have a disproportionate amount of non-protocol processing - likely reducing the efficacy of RPS and RFS optimizations. This indicates that overlay networks may not be very compatible with existing flow parallelization methods.

Figure 6 shows the normalized average bitrates across the different optimizations and replica counts. One can make the following observations: (1) The 95% confidence interval shrinks as the number of replicas increases. (2) The normalized bitrate resulting from RPS, RPS+, RFS, RFS+ dips going from 1 replica to 2 replicas, then improves as the number of replicas increases. Having only 2 replicas does not stress the server's CPU enough to see significantly high performance benefits from parallelized flows, therefore possibly exacerbating the slight software overhead that RPS incurs, leading to slightly decreased performance. (3) RPS and RPS+ optimizations improve the average bitrate over the baseline setup, except in the case of 2 replicas; while RFS and RFS+ improve the average bitrate over the baseline setup in all scenarios. (4) RPS performs better than RPS+ for the 1 and 2 replicas cases, while RPS+ performs better than RPS when using more replicas. The same observation holds for RFS performance compare with RFS+. (5) RFS+ leads to the best normalized average bitrate when the number of replicas is more than or equal to 4 (the number of cores), while RFS leads to the best normalized average bitrate when the number of replicas is less than the number of cores. Even though RFS is based on RPS, the extra cache locality benefits are the reason for the increased performance.

VI. Conclusion and Future Work

In a VXLAN-based overlay network, our work demonstrates that the largest bitrate improvements are seen when Receive Flow Steering (RFS) is enabled at the host and container levels, reaching an average increase of up to 24% over the baseline case. Enabling these optimizations on only the host shows marginal improvements in bitrate around $8-9\%$ over the baseline. Generally, more replicas provide greater benefits from RPS and RFS optimizations. However, due to overlay networks having a disproportionate amount of non-protocol level network processing, RPS and RFS are not able to exploit flow parallelism to its full potential. These findings show that for high throughput applications with frequent inter-host communication patterns, enabling RFS or RPS at the host and container levels could improve performance.

In order to enable these optimizations at the container-level, privileged container access is required in order to remount the `/sys` filesystem with read-write permissions, since containers have this filesystem as read-only by default. This introduces security implications if an adversary is able to gain access to the container for example. We intend to investigate these as a future endeavor.

Furthermore, we plan on extending our research along different fronts, such as (1) exploring how RPS and RFS optimizations perform for UDP workloads since RPS and RFS compute hashes differently based on the protocol used, which could affect the behavior of how packets are distributed to different cores; (2) studying the efficacy of flow-parallelization optimizations when using different encapsulation protocols; (3) evaluating optimizations in CNIs that do not rely on overlay networking; (4) examining the power tradeoff of these optimizations against throughput gains; (5) evaluating tail latency as a result of these steering techniques.

References

[1] "Kubernetes Documentation," Kubernetes, 20-Apr-2024. [Online]. Available from: https://kubernetes.io/docs/home/ 2026.03.08

[2] "OpenShift Container Platform 4.21", Red Hat Documentation. [Online]. Available from: https://docs.redhat.com/en/documentation/openshift_container_platform/4.21 2026.03.08

[3] CNI. [Online]. Available from: https://www.cni.dev/ 2026.03.08

[4] C. Study, "Cloud Native Computing Foundation," CNCF, 15-Nov-2024. [Online]. Available from: https://www.cncf.io/ 2026.03.08

[5] "About Calico," Calico Documentation. [Online]. Available from: https://docs.tigera.io/calico/latest/about/ 2026.03.08

[6] "Cloud Native, eBPF-based Networking, Observability, and Security," Cilium. [Online]. Available from: https://cilium.io/ 2026.03.08

[7] Flannel-Io, "flannel," GitHub. [Online]. Available from: https://github.com/flannel-io/flannel 2026.03.08

[8] J. Pelletier, "2024 Kubernetes Benchmark Report: The Latest Analysis of Kubernetes Workloads," CNCF, 26-Jan-2024. [Online]. Available from: https://www.cncf.io/blog/2024/01/26/2024-kubernetes-benchmark-report-the-latest-analysis-of-kubernetes-workloads/ 2026.03.08

[9] K. Suo, Y. Shi, A. Lee, and S. Baidya, "Characterizing networking performance and interrupt overhead of container overlay networks," Proceedings of the 2021 ACM Southeast Conference, pp. 93–99, Apr. 2021.

[10] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An analysis and empirical study of Container Networks," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Apr. 2018.

[11] "8.6. receive-side scaling (RSS)" 8.6. Receive-Side Scaling (RSS) — Red Hat Product Documentation. [Online]. Available from: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-rss 2026.03.08

[12] J. Corbet, "Receive packet steering," Receive packet steering [LWN.net], 17-Nov-2009. [Online]. Available from: https://lwn.net/Articles/362339/ 2026.03.08

[13] J. Edge, "Receive flow steering," Receive flow steering [LWN.net], 07-Apr-2010. [Online]. Available from: https://lwn.net/Articles/382428/ 2026.03.08

[14] "The linux kernel," Scaling in the Linux Networking Stack - The Linux Kernel documentation. [Online]. Available from: https://docs.kernel.org/networking/scaling.html 2026.03.08

[15] J. Lei, M. Munikar, H. Lu, and R. Jia, "Accelerating packet processing in container overlay networks via packet-level parallelism," 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 79–89, May 2023.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

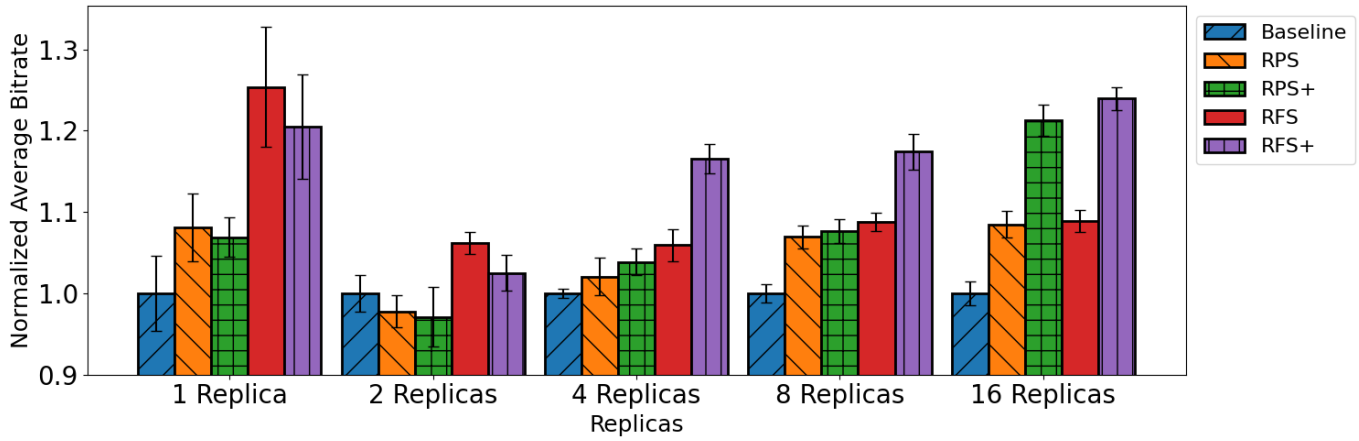


Figure 6. Normalized average bitrate for different number of replicas.

[16] S. Novianti and A. Basuki, "The performance analysis of Container Networking Interface Plugins in Kubernetes," 6th International Conference on Sustainable Information Engineering and Technology 2021, vol. 9, pp. 231–234, Sep. 2021.
[17] Weaveworks, "Weaveworks/weave," GitHub. [Online]. Available from: https://github.com/weaveworks/weave 2026.03.08
[18] "Overlay Network Driver," Docker Documentation. [Online]. Available from: https://docs.docker.com/engine/network/drivers/overlay/ 2026.03.08
[19] J. Lei, K. Suo, H. Lu, and J. Rao, "Tackling parallelization challenges of kernel network stack for Container Overlay Networks," USENIX, 01-Jan-1970. [Online]. Available from: https://www.usenix.org/conference/hotcloud19/presentation/lei. 2026.03.08
[20] F. Lin, X. Zhang, G. Chen, L. Chen, K. Li, and H. Jiang, "Slim and fast: Low-overhead container overlay network with fast connection setup," IEEE Transactions on Cloud Computing, vol. 12, no. 1, pp. 1–12, Jan. 2024.
[21] Y. Ma, S. Smith, B. Dai, H. Franke, B. Sukhwani, S. Asaad, J. Xiong, V. Kindratenko, and D. Chen, "UNINET: Accelerating the Container Network Data Plane in IaaS clouds," 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), vol. 33, pp. 115–127, Jul. 2024.
[22] "What is Tunneling?," Cloudflare. [Online]. Available from: https://www.cloudflare.com/learning/network-layer/what-is-tunneling/. 2026.03.08
[23] "What is VXLAN?," Juniper Networks. [Online]. Available from: https://www.juniper.net/us/en/research-topics/what-is-vxlan.html 2026.03.08
[24] "What is GRE Tunneling?," Cloudflare. [Online]. Available from: https://www.cloudflare.com/learning/network-layer/what-is-gre-tunneling/ 2026.03.08
[25] "5.2. IP in IP Tunneling," IP in IP Tunneling. [Online]. Available from: https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.tunnel.ip-ip.html 2026.03.08
[26] "etcd," etcd, 22-Mar-2024. [Online]. Available from: https://etcd.io/ 2026.03.08
[27] J. Lei, M. Munikar, K. Suo, H. Lu, and J. Rao, "Parallelizing packet processing in container overlay networks," Proceedings of the Sixteenth European Conference on Computer Systems, Apr. 2021.
[28] S. Vasudevan, "srinva/accel-overlay-NW," GitHub. [Online]. Available from: https://github.com/srinva/accel-overlay-nw 2026.03.07
[29] "IPerf," iPerf.fr. [Online]. Available from: https://iperf.fr/iperf-doc.php 2026.03.08
[30] "MPSTAT(1): Report processors related statistics," Linux Man Page. [Online]. Available from: https://linux.die.net/man/1/mpstat 2026.03.08

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

The Final Frontier of Orchestration: Bringing Kubernetes to On-Field and Embedded Devices Beyond Cloud-to-Edge Continuum

Pallav Kumar Deb, Jorge Carola, Susmit Shegokar, Wolfgang Forstmeier
Siemens Technology and Services Private Limited,
Germany
e-mail: (pallav.deb, jorge.carola, susmit.shegokar, wolfgang.forstmeier)@siemens.com

***Abstract*—The resource demands of cloud-native orchestration tools make it challenging to include last-mile devices, particularly embedded and on-field systems, in Cloud-to-Edge continuum deployments. This represents the final missing piece in achieving truly end-to-end orchestration across federated environments. This paper introduces a federated architecture that extends Kubernetes (K8s)-native orchestration beyond the Cloud-to-Edge continuum, reaching deeply embedded on-field devices such as actuators powered by single-board processors commonly found on industrial floors and in smart buildings. While K8s allows custom shims by leveraging WebAssembly (WASM) runtimes, it is not yet ready to be used on the constrained on-field devices due to its desired memory footprint. This paper introduces a splitted shim, built on the WASM Micro Runtime (WAMR). It decouples the runtime into a lightweight client-side agent and a proxy shim that can be deployed either on an edge node or in the cloud as a service, depending on deployment requirements. This proxy mediates communication with the K8s control plane, enabling orchestration without requiring Open Container Initiative (OCI) bundle support on the constrained device. The architecture is OS- and platform-independent, supports heterogeneous environments, and drastically reduces the resource footprint, enabling scalable, cloud-native orchestration from centralized cloud infrastructure to edge gateways and minimalistic field devices.**

Keywords-cloud–edge–field continuum; orchestration; kubernetes; webassembly (WASM); constrained devices

I. INTRODUCTION

Workload migration is a critical enabler in modern distributed computing environments to seamlessly move applications across heterogeneous infrastructures for optimized performance, resilience, and resource utilization. With the advent of Docker and its salient features, industries have widely adopted containerized architectures. Consequently, platforms such as Kubernetes (K8s) have become the de-facto standard for orchestration and managing workloads at scale. It offers powerful and clean abstractions for deployment, scheduling, monitoring, and service management. Its level of simplification, consistency, and automation has led to undisputed adoption across cloud, edge, and hybrid computing deployments.

K8s depends on key components, such as the Application Programming Interface (API) server, etcd (data store), scheduler, controller manager, kubelet (node agent), and kube-proxy (networking). Together, these components make K8s reliant on resource-capable devices, which confines its suitability to only relatively powerful edge devices and not those operating at the last mile, referred to hereafter as on-field devices. These devices are typical in industrial and field deployments. However, due to their tiny CPUs, limited memory, and ruggedized hardware, running the standard K8s stack is a challenge. For instance, in popular use cases such as environmental sensing, motion and inertial monitoring, biometrics and health data acquisition, distance and proximity measurement, touch and interaction sensing, and similar scenarios, STM-series microcontrollers are the primary units responsible for collecting, processing, and transmitting data. They typically have flash memory (up to 4 MB for high-end series) for code storage and Static Random-Access Memory (SRAM) (up to 1.5 MB) for data handling, which makes them unsuitable for running K8s components. While there are lighter alternatives to K8s available, they are still not ready to be hosted on constrained devices. Table I highlights some of the popular K8s distributions and their hardware requirements [1]. As a result, many real-world systems end up excluding these constrained devices from orchestration entirely, managing them instead with custom scripts or proprietary solutions that do not integrate well with modern cloud-native workflows, eventually limiting interoperability and scalability.

TABLE I. HARDWARE REQUIREMENTS FOR KUBERNETES DISTRIBUTIONS

Environment	Minimum RAM	Minimum CPU	Disk Space
K8s	4 GB	2 cores	20 GB
K3s	512MB	1 core	200MB
Minikube	2GB	2 cores	20GB
Kind	1GB	1 core	5GB
MicroK8s	540MB	1 core	300MB

In this work, we aim to challenge the long-standing assumption that last-mile devices must remain excluded from the K8s ecosystem and isolated from cloud-native capabilities. To address this gap, we propose an approach that allows resource-constrained devices to be onboarded into K8s using off-the-shelf solutions. For brownfield deployments, our method requires only minimal southbound changes, allowing these constrained devices to appear directly on K8s dashboards without requiring any northbound modifications. In our design, the control plane remains agnostic to the underlying limitations of these devices, treating them as standard nodes within the cluster. This enables new opportunities for workload migration, dynamic reconfiguration, and adaptive control from cloud to edge and all the way to on-field devices.

Use Case: Smart-building sensors, such as smoke detectors run on constrained devices. They stay idle most of the time and cannot join orchestration frameworks due to reasons mentioned earlier. Currently, workloads are manually mapped one-to-one.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

The proposed solution enables simple, cloud-native workload orchestration without changing existing models.

A. Motivation

With the advent of WebAssembly System Interface (WASI), WebAssembly (WASM) workloads run outside of the web browser and directly on devices using POSIX-like interfaces [2]. This enhances portability, strict isolation, near-native speed, and cross-language composability. Containerd [0] has started supporting docker-based WASM images by introducing runtimes to their *shims* in contrast to Linux-based shims [3]. A containerd shim is a lightweight daemon that manages the container's lifecycle. This has allowed K8s to also support WASM-based workload orchestration. WebAssembly Micro Runtime (WAMR) is a popular choice for running WASM workloads on constrained devices. However, the containerd shim coupled with the WAMR runtime is ~10 MB, which is not suitable for constrained devices (refer Section I). In addition to the runtime, the workloads have Open Container Initiative (OCI) relevant files included in the images, which further demands space. Such memory requirement alone restricts installation of the K8s components, which is the main motivation of our work. We propose ***Federated Shim*** as a method to onboard constrained device by splitting the shim into 2 components. A proxy component resides on an edge device (or optionally on the cloud for non air gapped devices) and handles K8s operations. The executor resides on the constrained device which is responsible for running the workload. Details about the strategy are available in Section III.

B. Contributions

In this work, we split the containerd shim into 2 components for making them more compatible for constrained devices. This makes them suitable for onboarding to K8s in a cloud-native fashion. Towards this, we make the following contributions:

- ***Federated shim architecture***: We introduce a two-part WASM shim that offloads container-runtime responsibilities to an edge-side proxy while leaving only a lightweight execution agent on the constrained device. This eliminates the need for OCI bundles or containerd on the field device.
- ***Platform-independence***: We reduce the WAMR footprint to ~30 KB through custom builds and Ahead-of-Time (AOT) compilation, enabling devices with only a few hundred kilobytes of RAM to run K8s-orchestrated workloads.
- ***Seamless deployment***: We enable constrained devices to appear as regular K8s nodes using standard dashboards and tooling, requiring no northbound modifications, making it suitable for brownfield deployments.
- ***Status reporting and observability***: We implement a message-driven mechanism that enables sharing logs, health status, and execution results to the K8s control plane, ensuring observability despite device limitations.
- ***Modular, plug-and-play components***: We design a fully modular system. The communication module, execution runtime, proxy layer, and onboarding components are independent and replaceable without affecting the rest of the stack, ensuring portability across heterogeneous hardware and simplifying integration into diverse industrial deployments.

The remainder of this paper is structured as follows. In Section II, we review related work and existing approaches. Section III presents the proposed federated shim architecture. Section IV outlines the experimental setup, and Section V discusses the key observations and performance results. Finally, Section VI concludes the paper and highlights future work.

II. Related work | Methods

The literature frequently cites the vision of a unified compute continuum stretching from massive cloud clusters down to tiny embedded devices, which consistently breaks down at the far edge. K8s has earned its place as the dominant orchestration platform for cloud-native workloads, offering declarative scheduling, self-healing controllers, and a mature industrial ecosystem [4]. The problem is that its architecture was never designed with microcontroller unit (MCU) in mind. A Class 2 MCU typically offers 256 KB of RAM and 256 KB of Flash. Even a stripped-down Kubelet agent consumes hundreds of megabytes at steady state, which is not suitable for constrained hardwares [6][7].

The community has responded with progressively lighter distributions. Goethals *et al.* [6] showed with FLEDGE that a Virtual Kubelet bridge could bring K8s scheduling to low-resource Linux edge nodes like the Raspberry Pi, side-stepping the full control plane while keeping standard kubectl workflows intact. K3s took a different angle, collapsing every control-plane component into a single binary below 100 MB and substantially reducing memory consumption relative to upstream K8s, landing at a practical minimum of around 512 MB RAM on a combined server–agent node [12]. KubeEdge pushed the agent footprint lower still, to on the order of 70–100 MB depending on configuration, by splitting responsibility between a cloud-side Edge Controller and a lean EdgeCore daemon on the device, and adding Message Queuing Telemetry Transport (MQTT)-based IoT messaging through its built-in Event Bus [7]. Taken together, these works prove that the K8s control plane can be cleanly decoupled from its execution environment, a principle that Federated Shim deliberately exploits, but carries all the way down to the MCU.

On the runtime side, the WASM ecosystem has made equally important progress. Wallentowitz *et al.* [13] systematically benchmarked the major WASM runtimes against embedded targets and concluded that WAMR and Wasm3 are the two most viable candidates for microcontroller deployment. WAMR offers the richer feature set AOT and Just in Time (JIT) compilation. Wasm3 trades those features for a smaller interpreter footprint. Mislav *et al.* [14] validated this further with empirical evaluation across the Raspberry Pi Pico, ESP32-C6, and nRF5340, finding that both runtimes deliver acceptable cross-platform portability and execution speed even under strict resource budgets, suggesting that WASM is approaching practical viability for production IoT scenarios and not merely a research curiosity. Comparatively, there are approaches for packaging WASM modules inside standard OCI images to

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE II. COMPARISON WITH EXISTING SOLUTIONS.

Methods	K8s	WASM Runtime	Workload Migration	OSI Support	OS Support	Constrained Devices	Minimum Hardware	Messaging/ Network
Vaño *et al.* [4]	✓	Partial	✓	✓	Linux	Partial	Edge-class	–
Feather [5]	✓	Wasmtime	–	✓	Linux	✓	∼86 MB RAM	Kubernetes API
FLEDGE [6]	✓	–	✓	✓	Linux	✗	∼60 MB RAM*	OpenVPN Overlay
KubeEdge [7]	✓	–	✓	✓	Linux	✗	70 MB (agent) / 256 MB (device)	MQTT (EventBus)
Tinto *et al.* [8]	–	✓	✓	–	Heterogeneous	✓	Embedded-level	Bytecode Serialization
wasmCloud [9]	✓	Wasmtime	–	✓	Linux / Bare	✓	Embedded nodes	NATS
Krustlet [10]	✓	Wasmtime	–	Partial	Linux	✗	Std PC / Edge	Kubelet API
Ocre [11]	–	✓	–	Partial	RTOS (Zephyr-class)	✓	Embedded-class	REST / Overlay
Federated Shim	✓	**WAMR**	✓	✓	**None (RTOS/Bare)**	✓	**10 KB RAM / MCU**	**MQTT**

*Approximate value; exact minimum hardware requirement not explicitly stated in the paper. Krustlet is no longer actively maintained [10].

maintain registry compatibility [3]. However, they reintroduce the abstraction layers of container runtime, namespace isolation, and image-layer unpacking, that WASM was originally chosen to eliminate.

A meaningful gap persists between what the literature calls lightweight and what a deeply constrained device actually needs. Most solutions that claim a small footprint still assume a Linux kernel and tens of megabytes of RAM [6]. Vaño *et al.* [4] highlighted in their survey that stateful migration mechanisms across the edge cloud continuum are frequently incompatible with OCI registries, which cuts them off from standard cloud-native tooling and forces operators to maintain separate pipelines. Device onboarding is an equally neglected dimension. Lacalle *et al.* [4] observed that most edge frameworks rely on manual provisioning steps that do not scale gracefully to fleets of volatile or intermittently connected hardware. Projects such as Ocre [11] have tried to address the hardware side of this problem directly, bringing container-like semantics to RTOS-class devices through a WASM execution layer, a REST management API, and support for hardware as modest as an Arm Cortex-M3. However, as Goethals *et al.* [6] would recognize, a runtime without an orchestration plane is only half a solution. Ocre does not natively integrate with K8s and provides no mechanism for stateful migration between nodes.

Synthesis: The Federated Shim for embedded devices is motivated directly by the gap that each of these works leaves open. We take the Virtual Kubelet abstraction that Goethals *et al.* [6] demonstrated at the Linux edge and extend it all the way to bare-metal microcontrollers, combining it with MQTT-based zero-touch onboarding so that a physical device registers itself as a standard virtual node without any manual configuration step. A federated shim layer then routes scheduled workloads through containerd into the WAMR runtime [13], which is the same runtime that Wallentowitz *et al.* [13] benchmarked as the most capable embedded WASM engine enabling OCI-compatible, stateful WASM on hardware with only tens of kilobytes of RAM. The result is a single orchestration fabric that finally reaches the on-field devices rather than stopping at the edge, and does so without abandoning the standard K8s APIs that the rest of the stack already depends on. We highlight the key differences of the existing solutions against the proposed federated shim in Table II.

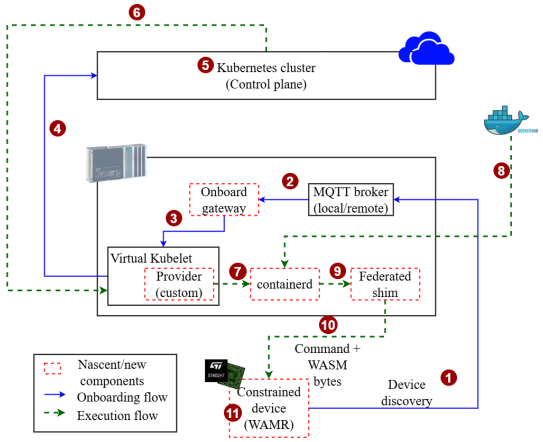


Figure 1. Information flow of the Federated Shim.

III. SYSTEM ARCHITECTURE

We introduce a modular split in the coventional containerd WASM shim to support deployment across heterogeneous environments (refer Figure 1). The edge device serves as a proxy layer, hosting the task service that interfaces with containerd. This service manages lifecycle events and creates WASM instances. Upon instance creation, the edge device forwards execution requests to a constrained device, which hosts the WASM runtime. According to original design, the runtime is invoked through the Youki library, a Rust-based container runtime, enabling secure and efficient execution of WASM workloads. This separation allows the edge device to handle orchestration and communication overhead, while the constrained device focuses solely on lightweight execution.

We adopt WAMR as our choice of runtime as it is tailored for environments with limited memory and compute capacity. While the default WAMR shim build is around 9.2 MB, our target devices have only 500 KB of RAM, requiring significant optimization. A key architectural decision was to eliminate OCI components from the constrained device. These components, typically used for container lifecycle and image management, introduce unnecessary overhead in minimal environments. By removing them, we not only simplify the runtime footprint but also eliminate the dependency on the Rust-based Youki library. Consequently, we focus solely on WAMR, which is written in C. The removal of these primary components does not affect functionality as we introduce agents on both the edge and

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

constrained devices. Compared to the OCI components, the designed agents have significantly low footprint. We further reduce the WAMR runtime size to ~30 KB using custom build flags and AOT compilation. Further, by leveraging WAMR's modular and portable library design, we ensure platform independence and consistent behavior across diverse operating systems without requiring OS-specific adaptations.

The flow of the proposed deployment is as follows:

1) K8s control plane sends control commands to containerd.
   a) Based on deployment strategy, the control plane may be hosted on the edge or the cloud.
2) Containerd forwards to proxy manager.
   a) We use Virtual Kubelet to enable proxy.
3) Proxy manager interacts with the constrained device using a communication module.
4) Communications between the edge device consists of the command along with the WASM bytes.
5) Receiver in the constrained device passes the command and WASM bytes to an execution manager.
6) Execution manager initiates the WAMR runtime and executes the workload.
   a) In the case of multiple workloads, threading is applied as the constrained devices are typically single core processor boards.
7) Based on device type and design requirements, logs of the executions are either stored in the local file system or transferred to the edge device.
8) Monitoring and health status are also sent to the edge device.

The information flow consists of 2 modes: (i) Device onboarding and (ii) Workload execution. As shown in Figure 1, the information flows for each work as follows:

Device onboarding: 1. Constrained device sends a self-discovery message to the edge device. It leverages an MQTT broker (hosted locally/remotely) for sharing messages. 2. The broker forwards the message to an onboarding gateway. 3. The gateway spins up a virtual kubelet, which acts as proxy for the constrained device. 4. Virtual kubelet registers the constrained device on the control plane. 5. K8s control plane registers the device and mandates the use the Federated Shim.

Workload execution: 6. K8s control plane assigns a workload/pod to the constrained device, which is passed to Virtual Kubelet. 7. Virtual kubelet's custom provider triggers containerd. 8. Containerd fetches the pod-specific OCI bundles from the virtual kubelet. 9. Containerd then calls the federated shim interface for the constrained device. 10. Lifecycle commands and WASM bytes are sent to the constrained device. 11. Constrained device executes the workload.

IV. Experiment Setup

In this Section, we present our experiment setup, which mimics real industrial cloud–edge–field environments. We use a Raspberry Pi 4 Model B as the constrained on-field device and a Siemens SIMATIC IPC427E [15] as the edge node. We host the K8s control plane in AWS cloud. The Raspberry Pi 4B is an ARM-Cortex-A72-based microcontroller board with 4 cores, running at 1.8 GHz and 4 GB RAM. It supports Ethernet, dual-band Wi-Fi, and Bluetooth connectivity. While this is not our intended target constrained device, we show consumption metrics in Section V, which proves the suitability of running the proposed solution on the former. The Siemens IPC 427E, in contrast, is an industrial-grade edge system and has sufficient resources to host the proxy components like containerd, Virtual Kubelet, and the proposed federated proxy layer. We choose addition of 2 numbers as our WASM evaluation workload which returns back the sum. Our bias towards this workload is due to its simplicity. It helps us in keeping the computation time negligible (as WASM optimization is beyond the scope of this work) and we primarily focus on the federated shim metrics with ease. Additionally, this workload covers key runtime operations like argument passing across the proxy to on-field device, execution inside a sandboxed module, and retrieval.

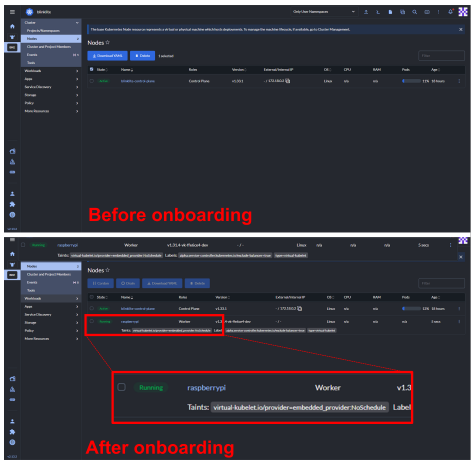


Figure 2. Onboarding the constrained device to K8s (Rancher dashboard).

V. Observations

In this section, we present our observations while deploying the proposed Federated Shim. Since the proxy component can be deployed either on capable edge devices or on the cloud, we focus primarily on the constrained devices. They represent the most critical bottleneck in extending K8s orchestration to last-mile deployments.

A. Device Onboarding

We choose Rancher [16] as our K8s management platform. It is open-source, enterprise-grade, and enables deployment, management, and security for containerized applications across diverse environments. It also simplifies multi-cluster operations by providing a single centralized management console. We validate the onboarding workflow of the proposed solution by monitoring the available nodes in the cluster displayed in the Rancher dashboard. Prior to onboarding, the node list is empty (refer Figure 2 top). Once the constrained device makes the onboarding call, the device appears as a fully registered node (refer Figure 2 bottom). This demonstrates that the proposed Federated Shim onboards the constrained device to the K8s control plane without requiring any northbound modifications.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

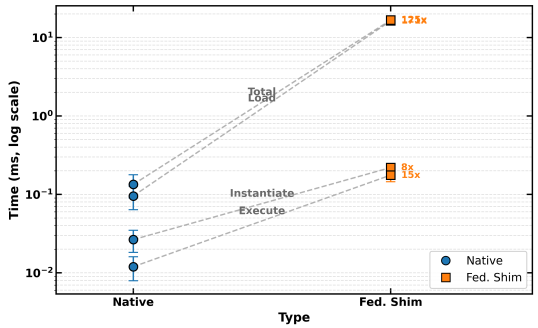


Figure 3. Time consumption in different phases of execution.

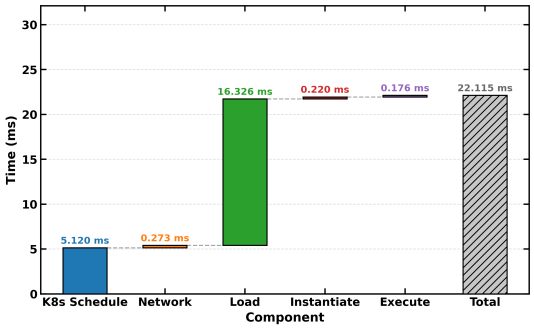


Figure 4. End-to-End Time Consumption.

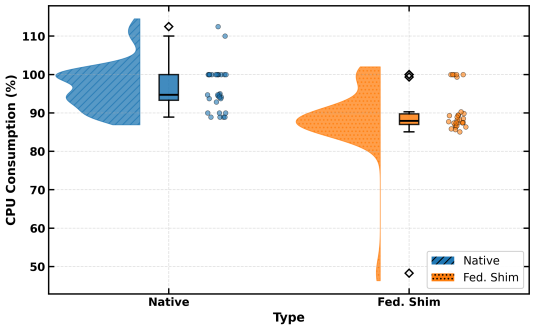


Figure 5. CPU Consumption.

B. Execution Time

In Figure 3, we present the execution times in log scale for better representation. We observe that the total time required for executing the workload by the federated shim is $125x$ compared to native WASM (on average, 16.72 ms and 0.134 ms, respectively). This raises concerns and requires a deep dive. Consequently, we breakdown the time into 3 phases. The *Load* phase reads the WASM binary file from storage into RAM and parses its structure. *Instantiate* phase takes the loaded module and allocates its linear memory, initializes global variables, and wires up all import/export bindings. *Execute* phase hands the fully initialized WASM module directly to the CPU interpreter (WAMR in this case) and runs the actual program logic. We observe that the load phase is the most expensive of all for the federated shim. We attribute this behaviour to requiring disk input/output, deserializing, and initializing WASM. While the instantiation and execution overheads are relatively contained, the federated shim induces overhead when compared to native phases by $8x$ and $15x$, respectively. This observation has been consistent in our experiments suggesting that the federated shim incurs per-invocation cost.

Compared to native execution, the device is advocating MQTT polling and other operations (both OS and shim) while also making sure that the workload executions happen. The differences in the instantiate and execution phases may be occuring due to context switching. However, the higher overhead of loading all components reflects additional marshalling and inter-layer communication costs of the shim rather than fundamental computational inefficiency, which can be improved with modifications.

C. End-to-End Time Consumption

To give a perspective of the scheduling, networking, and data transfer overheads, we decompose the end-to-end invocation latency into 5 components (refer Figure 4). We observe that the total time required to schedule and get results is ~22 ms, of which the actual workload execution time is ~16 ms (refer Section V-B). The K8s scheduling is another major contributor to the time, which is almost 23%. We attribute this to the operations in the control plane for dispatching the workload through the proxy shim, which is using gRPC protocol. This overhead is typical to K8s is independent of the proposed solution and would uniformly apply to any other container runtime shims. On the other hand, the transit time is only 1.2%, demonstrating that the MQTT protocol adds negligible latency. However, we do plan to replace this with protocols that are more suitable for constrained and intermittent networks like nanoIP, CoAP, or others.

We infer from the breakdown the total latency is dominated by the workload execution phases and K8s scheduling, implying that the communication protocols are not a limiting factor. Since the time due to K8s scheduling is a cluster-level byproduct, the workload execution phases contributing ~75% of the time need attention and will be our focus for improvement.

D. CPU Consumption

In Figure 5, we present the CPU consumption during our experiments. Interestingly, we observe that the proposed federated shim, on average, consumes less CPU than native. On the other hand, we observed in Figure 3 longer in wall clock time during execution. With InterQuartile Range (IQR) of 6.69%, the native distribution is tight and uniform, with occasional more than 100% due to compute bursts. The Federated Shim's IQR is even narrower (2.77%), implying that most runs are stable around 88% consumption. The outlier of 48.3% inflates its standard deviation. We conclude that while the runs are consistent, a subset of the executions experience severe CPU under-utilization, which may be caused by prolonged blocking on read/write operations. These non-deterministic scheduling delays yields the CPU entirely.

We infer that the federated shim's overhead is input/output bound, rather than compute-bound, making the IPC communi-

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE III. OVERHEAD SUMMARY

Phase	Time (×)	Mem. (ΔKB)	CPU (Δ%)
Load	171×	+188	+1.5
Instantiate	8×	+84	+0.5
Execute	15×	+8	-7.6

cation channel (or deployment strategy) as our primary target for improvement and not the WASM runtime.

E. Overhead Summary

We previously observed that the federated shim introduces overhead in execution time. We take a closer look at the memory consumption and then highlight the collective overheads in Table III. The memory consumption follows a similar trend as the execution time in the Load phase by $+188$ KB, we attribute this to the per-invocation IPC channel setup and module binary transfer to the worker process. The instantiation and execution add smaller overheads ($+84$ KB and $+8$ KB, respectively). This may be due to the cross-process linear memory allocation.

We infer that while the execution times and memory allocations are more for the federated shim, the CPU utilization follows an opposite trend. As most clock time is spent blocking on IPC reads rather than active computation (refer previous sections), we reaffirm the IPC channel and deployment strategy as our primary optimization target.

VI. CONCLUSION AND FUTURE WORK

In this work, we addressed the inclusion of embedded and resource-constrained on-field devices in K8s managed ecosystems, which is final missing piece for cloud-edge-on-field orchestration. Existing K8s stack, inclusive of WASM shim support cannot run directly on such single-board industrial microcontrollers due to their resource demands and dependency on OCI bundle support. To bridge this gap, we split the shim architecture and built on the WebAssembly Micro Runtime (WAMR) runtime. This decouples the K8s shim into a lightweight client-side agent, and deployed on the constrained device. The proxy shim is deployed on an edge node or can be deployed as a cloud service. It mediates all interaction with the K8s control plane. Additionally, with the use of strategic techniques, we achieve OS independence and the proposed solution works wherever WAMR is installed, increasing the range of device. Consequently, the device remains agnostic to the orchestration complexity. Through real-world experimentation, we showed the feasibility of the proposed solution and its hardware consumption.

In the future, we plan to extend this work as a derivative of the observations in Section V. We observed that the high execution times and memory consumption are a result of heightened input/output operations and IPC operations, in contrast to compute and MQTT protocol. This necessitates the improvement in the split shim and deployment strategies. We will also be extending this solution to support more (WASM and non-WASM) runtimes and communication protocols.

REFERENCES

[1] A. Iyer, "The Ultimate Guide to Lightweight Kubernetes Environments", Accessed: 2026-02-10, Signadot, May 2025. [Online]. Available: https://www.signadot.com/articles/the-ultimate-guide-to-lightweight-kubernetes-environments

[2] W3C WebAssembly Community Group, "WASI: The WebAssembly System Interface", Accessed: 2026-02-10, WASI Subgroup, W3C WebAssembly Community Group, 2025. [Online]. Available: https://wasi.dev/

[0] containerd authors, *Containerd: An industry-standard container runtime with an emphasis on simplicity, robustness and portability*, Accessed: 2026-03-13, 2026.

[3] S. Cheng, "WebAssembly on Kubernetes: from containers to Wasm (part 01)", Accessed: 2026-02-10, Cloud Native Computing Foundation, Mar. 2024. [Online]. Available: https://www.cncf.io/blog/2024/03/12/webassembly-on-kubernetes-from-containers-to-wasm-part-01/

[4] R. Vaño, I. Lacalle, et al., "Cloud-native workload orchestration at the edge: A deployment review and future directions", *Sensors*, vol. 23, no. 4, p. 2215, 2023. DOI: 10.3390/s23042215

[5] M. Sebrechts, J. Van der Auwera, B. Volckaert, and F. De Turck, "Adapting Kubernetes Controllers to the Edge: On-Demand Control Planes Using Wasm and WASI", in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022, pp. 1–9.

[6] T. Goethals, F. De Turck, and B. Volckaert, "FLEDGE: Kubernetes compatible container orchestration on low-resource edge devices", in *Internet of Vehicles. Technologies and Services Toward Smart Cities*, ser. Lecture Notes in Computer Science, vol. 11894, Cham: Springer, 2019, pp. 215–230. DOI: 10.1007/978-3-030-38651-1_16

[7] KubeEdge Project, "KubeEdge: Kubernetes Native Edge Computing Framework", 2024, Accessed: Feb. 20, 2026. [Online]. Available: https://kubeedge.io

[8] E. Tinto, "Time-Predictable Runtime Infrastructure for the Edge-Cloud Continuum: Live Migration of Software Components Across Heterogeneous Nodes", M.S. thesis, University of Padova, 2024.

[9] wasmCloud Project, "wasmCloud Documentation", 2024, Accessed: Feb. 20, 2026. [Online]. Available: https://wasmcloud.com/docs/

[10] M. Fisher and M. Butcher, "Introducing Krustlet, the WebAssembly Kubelet", 2020, Accessed: Feb. 20, 2026. [Online]. Available: https://deislabs.io/posts/introducing-krustlet/

[11] LF Edge Project, "Ocre: Open Container Runtime for the Edge", 2023, Accessed: Feb. 20, 2026. [Online]. Available: https://lfedge.org/projects/ocre/

[12] Rancher Labs / SUSE, "K3s: Lightweight Kubernetes", 2024, Accessed: Feb. 20, 2026. [Online]. Available: https://docs.k3s.io/

[13] S. Wallentowitz, B. Kersting, and D. M. Dumitriu, "Benchmarking WebAssembly for Embedded Systems", *ACM Transactions on Architecture and Code Optimization*, pp. 1–21, 2025. DOI: 10.1145/3736169

[14] M. Has, T. Xiong, F. Ben Abdesslem, and M. Kusek, "WebAssembly on Resource-Constrained IoT Devices: Performance, Efficiency, and Portability", 2025. arXiv: 2512.00035 [cs.PF].

[15] Siemens AG, "SIMATIC IPC427E Industrial Edge Device", Accessed: 2026-02-20, Siemens, 2024. [Online]. Available: https://www.dex.siemens.com/edge/manufacturing-process-industries/simatic-ipc427e-industrial-edge-device

[16] Rancher by SUSE, "Rancher manager documentation", Accessed: 2026-02-10, 2024. [Online]. Available: https://ranchermanager.docs.rancher.com/

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

A Cloud-Native Architecture for Human-in-Control LLM-Assisted OpenSearch in Investigative Settings

Benjamin Puhani*, Kai Brehmer*, Malte Prieß†

*AI Research Unit of the
State Police of Schleswig-Holstein, Kiel, Germany
e-mail: {benjamin.puhani | kai.brehmer}@polizei.landsh.de

†Faculty of Computer Science and Electrical Engineering
Kiel University of Applied Sciences, Germany
e-mail: malte.priess@haw-kiel.de

***Abstract*—Complex criminal investigations are often hindered by large volumes of unstructured evidence and by the semantic gap between natural language investigative intent and technical search logic. To address this challenge, we present a design and feasibility study of a cloud-native microservice architecture tailored to private-cloud deployments, contributing to research in secure cloud computing and leveraging modern cloud paradigms under high security and scalability requirements. The proposed system integrates Large Language Models into a "Human-in-Control" workflow that translates natural-language queries into syntactically valid OpenSearch Domain-Specific Language expressions. We describe the implementation of a hybrid retrieval strategy within OpenSearch that combines BM25-based lexical search with nested semantic vector embeddings. The paper focuses on system design and preliminary functional validation, establishing an architectural baseline for future empirical evaluation. Technical feasibility is demonstrated through a functional prototype, and a rigorous evaluation methodology is outlined using the Enron Email Dataset as a structural proxy for restricted investigative corpora.**

Keywords-OpenSearch; Information Retrieval; Semantic Search; Investigative Settings; Enron Dataset.

I. INTRODUCTION

Investigations into international and transnational crimes, such as genocide, crimes against humanity, and related violations of international criminal and humanitarian law, require the analysis of large volumes of unstructured textual evidence, including witness statements, interview transcripts, and communication records. Proceedings under national universal jurisdiction frameworks, such as the German Code of Crimes against International Law (Völkerstrafgesetzbuch, VStGB), serve as a representative example of this broader class of investigations. Across such contexts, investigators face a recurring challenge: relevant evidence is often present in the data but remains difficult to access because of the gap between natural language investigative intent and the technical logic of search systems. Recent scholarship underscores this urgency: Skipanes et al. [1] identify the processing of unstructured text as a critical bottleneck in digital forensics and highlight that current methodologies largely fail to bridge the divide between computational opportunities and practical investigative reasoning.

Although modern search engines, such as OpenSearch [2], provide scalable indexing and retrieval capabilities, they inherently require input in a rigid and structured Query Domain-Specific Language (DSL) rather than in natural language. Most investigators and legal practitioners lack this expertise, leading them to rely on manual review or simple keyword searches. These approaches are poorly suited to capturing semantic variation, indirect references, variant spellings, and translation artifacts, and they limit recall - the ability to retrieve all relevant information - precisely in those cases where exploratory and hypothesis-driven search is required.

This paper addresses this semantic gap by presenting an exploratory proof-of-concept system that integrates Large Language Models (LLMs) as a translation layer between investigative intent and open search query logic. Natural language questions are mapped to syntactically valid OpenSearch DSL queries within a Human-in-Control architecture, in which the LLM functions as a supervised assistant rather than as an autonomous agent. The contribution of this paper is to outline a principal system design and methodological foundation, along with a novel architectural integration that provides a basis for future empirical evaluation. Because of legal and ethical constraints associated with real investigative data, the approach is demonstrated using the Enron Email Dataset [3] as a structural proxy that exhibits key characteristics of investigative corpora, including unstructured text, noisy data, and complex communication networks. Architecturally, the system is positioned within cloud-native computing paradigms, addressing challenges in private-cloud orchestration, horizontally scalable components, and secure cloud environments to ensure strict data sovereignty.

The remainder of this paper contextualizes this architecture within existing research (Section II), details the system design and hybrid retrieval strategy (Section III), demonstrates its functional feasibility (Section IV), and outlines the roadmap for empirical evaluation (Section V).

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

II. Related Work

To address the semantic and technical challenges of forensic data analysis, our work builds upon and integrates research from three primary domains.

A. Bridging the Semantic Gap in Digital Forensics:

In a recent comprehensive analysis, Skipanes et al. [1] identify the processing of unstructured text as a critical bottleneck in contemporary criminal investigations. They highlight that, while computational opportunities exist, current methods largely fail to bridge the gap between technical retrieval logic and the qualitative reasoning required by investigators. Our work directly addresses this architectural gap by operationalizing these opportunities within a secure on-premises environment.

B. LLM-Assisted Retrieval:

The integration of LLMs into Information Retrieval systems has evolved rapidly from simple re-ranking tasks to complex query generation [4]. Current approaches often focus on *Text-to-SQL* paradigms, in which LLMs translate natural language into structured SQL queries for relational databases [5]. However, these methods are inherently constrained by the unstructured and fuzzy nature of forensic text data. Conversely, Retrieval-Augmented Generation (RAG) grounds LLM responses in retrieved search results but may introduce hallucinations or lack the deterministic precision required for rigorous investigative filtering [6].

C. Cognitive Architectures and Prompting:

Our work builds upon the findings of Liu et al. [7] concerning the "Lost-in-the-Middle" phenomenon, which posits that LLMs often fail to identify relevant information in long contexts. We address this limitation by segmenting documents into semantic units rather than processing entire texts. Furthermore, we adopt the Chain-of-Thought (CoT) prompting strategy proposed by Wei et al. [8] to improve the logical consistency of generated OpenSearch queries. Unlike autonomous agents, our architecture enforces a "Human-in-Control" design that prioritizes the investigator's control over search logic to ensure procedural accountability within legal domains.

III. System Architecture and Methodology

To meet the high security and scalability requirements of law enforcement agencies, the proposed system is designed as a cloud-native microservice architecture. While leveraging modern cloud paradigms such as containerization and orchestration, the system is intended for deployment within a restricted private-cloud environment (e.g., on-premise Kubernetes) to ensure strict data sovereignty. At the same time, the architecture remains deployment-agnostic: the identical microservice stack can operate either in a fully orchestrated Kubernetes environment for large-scale installations or in a lightweight Docker Compose configuration for resource-constrained agencies. This flexibility stems from consistent containerization of all services and a clear separation between application logic and infrastructure orchestration, enabling the system to scale operational complexity according to organizational needs.

A. Cloud-Native Service Orchestration

The system follows a microservice architectural pattern composed of four distinct layers that communicate via RESTful APIs and asynchronous message queues (see Figure 1):

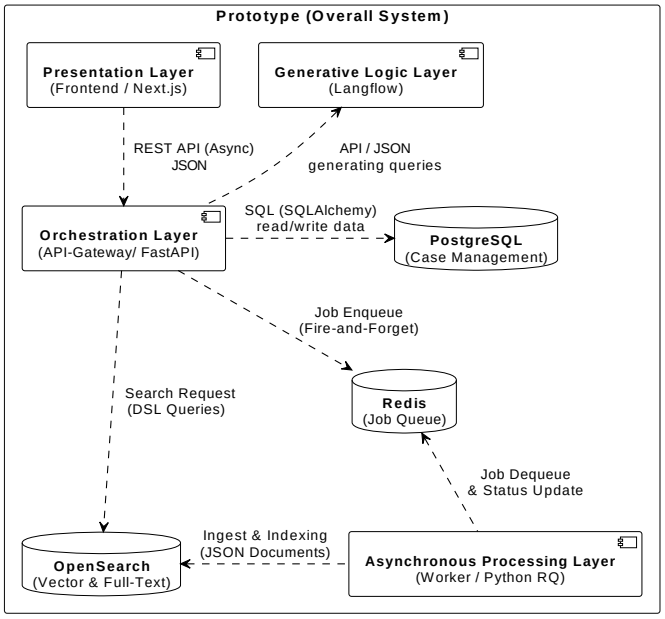


Figure 1. Schematic representation of the modular architecture and data flow.

Presentation Layer: A Single Page Application built with Next.js provides the investigative user interface. It uses Server-Side Rendering to optimize initial load performance and communicates asynchronously with the backend to ensure a non-blocking user experience, which is essential for reviewing large document sets.

Orchestration Layer (API Gateway): The core application logic is handled by a FastAPI backend. Unlike monolithic frameworks, FastAPI implements the ASGI standard, enabling native asynchronous request handling. This capability is critical for maintaining high throughput when coordinating I/O-intensive operations across the database, search engine, and LLM services. State management is offloaded to persistent stores - PostgreSQL for case management and Redis for job queues. Crucially, this design transforms the system from a stateless search engine into a case-management workspace. The backend tracks the "reviewed status" of each retrieved document, supporting a coverage-oriented workflow that enables investigators to systematically examine evidence by relevance.

Asynchronous Processing Layer (Worker): To handle large-scale data ingestion, we implemented a producer-consumer pattern using Redis. Python-based workers perform CPU-intensive heuristic parsing and disentanglement. However, computationally expensive vectorization is offloaded to the OpenSearch cluster thorugh a dedicated ingest pipeline that runs on specialized machine learning nodes. This design separates the cleaning logic from the inference workload, enabling independent scaling of ingestion workers and neural inference resources.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Generative Logic Layer: Instead of hardcoding prompt logic, the system integrates Langflow [9] as a visual low-code environment to manage interaction chains. To enforce data sovereignty, the architecture deliberately avoids reliance on public APIs. Instead, it uses locally hosted open-weight models (e.g., Llama 3 or Mixtral) running on on-premise inference servers via vLLM. This design ensures that no sensitive investigative intent leaves the secure private-cloud perimeter.

B. Semantic Segmentation and Ingestion Strategy

A major challenge in processing large-scale forensic data is the "Lost-in-the-Middle" phenomenon, in which LLMs fail to retrieve relevant information embedded in long, unstructured contexts [7]. Indexing a typical investigative document (e.g., a 50-page witness statement or an extended email thread) as a monolithic block degrades vector search performance because of the architectural token limitations of the underlying transformer models [10].

To address this limitation, we developed a configurable, modular adapter pattern within the asynchronous worker nodes. Unlike generic chunking strategies (e.g., fixed-size splitting), our system supports custom parsing profiles explicitly designed for each input type to preserve semantic boundaries in forensic data:

Heuristic Chunking (Legacy Documents): For unstandardized text documents, we implemented a custom regex-based heuristic to detect semantic shifts (e.g., speaker changes or timestamps). This approach allows the system to generate atomic segments even in the absence of structured separators.

Disentanglement (Communication Data): For the Enron dataset, the ingestion logic was tailored to disentangle forwarded message chains. By selectively stripping technical headers (e.g., `X-UID`) from the semantic payload (Level 2) while preserving them for structured filtering (Level 1), we minimize noise in the vector space.

This pre-processing step ensures that embeddings represent specific statements rather than diluted document-level averages.

C. Hybrid Data Modeling and Indexing

A core architectural decision was to implement a hybrid search index within OpenSearch. The current approach focuses exclusively on text-based data. Image and audio-based data would need to be converted into text. The hybrid search index addresses the fundamental linguistic limitations of purely lexical retrieval, specifically synonymy and polysemy [11]. By integrating vector-based semantic retrieval, our schema combines the strengths of both paradigms, a pattern applicable to both interview protocols and digital communication:

Unstructured Baseline (Level 1): The document's full text and metadata are indexed using standard BM25 lexical search [12]. This configuration ensures high recall for specific keywords, such as *names, case numbers, or senders/receivers.*

Structured Nested Embeddings (Level 2): To mitigate context dilution, we avoid embedding long documents as a single vectors. Instead, the atomic semantic units identified in Section III-B (e.g., specific paragraphs, message bodies, or individual statements) are stored as nested objects containing the segment text and a 384-dimensional vector embedding. We use the HNSW algorithm [13] with Cosine Similarity, adhering to the optimization objective of the underlying `paraphrase-multilingual-MiniLM-L12-v2` model [10]. To minimize architectural complexity, this model is provisioned through the OpenSearch ML Commons framework and executed directly on the cluster's internal ML nodes. This configuration ensures that ranking is determined by semantic alignment (vector orientation) rather than by vector magnitude. Additionally, a search-time synonym graph expands queries, e.g., mapping "detention" to "imprisonment", without increasing the physical index size.

This nested structure is designed to prevent "cross-object matching" errors in which a query matches unrelated parts of a document (e.g., matching a suspect's name from page 1 with an action described on page 10), and to enable the LLM to generate queries that target specific semantic segments.

Hybrid Fusion for Prioritized Review: During the exploratory evidence-review phase, minimizing *False Negatives* is paramount. Investigators require a ranking mechanism that surfaces the most relevant documents first to support the coverage-oriented workflow described in Section III-A. Our system employs a score normalization approach to fuse unbounded lexical scores (BM25) with normalized semantic scores (Cosine Similarity). This design ensures that a document describing "off-balance sheet debt" (semantic match) is ranked competitively with documents containing the specific project code "Raptor" (lexical match), even when their vocabularies do not overlap.

D. The "Human-in-Control" Generative Pipeline

The translation of natural language into the complex OpenSearch Query DSL is handled by a multi-agent LLM pipeline orchestrated via Langflow. To ensure domain agility without code modifications, the system employs schema-aware prompting: the current index definition is injected into the prompt context at runtime. Crucially, this design enforces a strict separation of concerns: the LLM operates exclusively on the abstract index schema, never on the actual evidentiary content. Unlike standard RAG workflows [14], which require feeding retrieved text into the model's context window, our architecture ensures that sensitive document payloads remain confined within the OpenSearch cluster and are never exposed to the inference context. Rather than relying on opaque "black box" logic, we implement a CoT workflow:

Reasoning & Generation: The first agent acts as a "Query Architect". It analyzes the user's intent and the injected schema structure, generating an intermediate reflection before constructing the JSON query.

Auditing (Quality Assurance): The second agent, the "Auditor", validates each generated query against known error patterns. For instance, it detects when the model attempts to search for structured entities (e.g., specific dates or person names) within the semantic vector field, which typically yields

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

lower precision. The Auditor enforces correct mapping to structured fields (e.g., moving a name search to the `sender` or `people` field) before execution.

Transparent Execution & Deterministic Retrieval: The validated query is then executed to provide immediate feedback. However, unlike fully autonomous agents, the system enforces transparency: the generated search logic (the "translation") is displayed alongside the results. Since all presented results are deterministic database retrievals rather than LLM-generated text, the risk of evidence hallucination is eliminated. The investigator retains full authority to evaluate the relevance of retrieved documents and iteratively refine the generated query logic, ensuring that the final assessment of evidence remains a human decision.

This pipeline ensures that the resulting DSL query is syntactically valid and semantically aligned with the investigator's intent prior to execution.

IV. Implementation Status and Preliminary Feasibility

The architecture described in Section III has been implemented as a fully functional prototype. The system successfully orchestrates interactions among the React frontend, the FastAPI gateway, and the asynchronous worker nodes. Initial functional tests confirm that the Langflow-based "Human-in-Control" pipeline is capable of generating syntactically valid OpenSearch DSL queries from natural language input. Specifically, the multi-agent setup (Generator and Auditor) demonstrated the ability to detect and correct basic logical errors, such as mapping named entities to incorrect fields, before execution. This technical readiness establishes the necessary baseline for the empirical evaluation strategy outlined in Section V, which will be the focus of subsequent research.

V. Conclusion and Future Work

This paper presented a cloud-native, microservices-based architecture designed to reduce technical barriers in accessing mass data for criminal investigations. By combining a hybrid OpenSearch index with a supervised LLM pipeline, we established a framework to translate investigative intent into high-precision database queries without compromising data sovereignty.

The next phase focuses on the quantitative calibration of the system. As real-world investigative data is legally restricted to operational use under strict purpose limitation regulations, it cannot be utilized for public academic benchmarking. Therefore, we will utilize the Enron email dataset [3] as a reproducible Ground Truth to simulate forensic retrieval tasks.

The primary objective is not merely to compare algorithms, but to determine the optimal hybrid configuration for forensic workflows, which prioritize high recall (coverage) over precision. The evaluation will specifically investigate two core architectural decisions:

Segmentation Granularity: Comparing retrieval performance when indexing monolithic documents versus the proposed granular segmentation (e.g., heuristic chunking or thread-splitting). We hypothesize that granular segments yield higher relevance scores for specific details but require aggregation to preserve context.

Score Fusion Tuning: Evaluating different weighting strategies for the Score Normalization (Lexical vs. Semantic weights) to maximize Recall@100. This metric, which measures the proportion of relevant documents retrieved within the top 100 results [11], is chosen to reflect the operational reality of investigators, who require the most critical evidence to appear within the first few pages of results.

To test these hypotheses, the evaluation design involves:

Adversarial Scenario Design: We will define 5–10 distinct search scenarios designed to simulate the semantic gap. Instead of searching for known identifiers (e.g., "Project Raptor"), queries will formulate abstract investigative intents (e.g., "conversations expressing anxiety about the company's financial stability" or "instructions to destroy documents"). These scenarios are specifically chosen to challenge lexical search engines, as they rely on sentiment and context rather than unique keywords.

Semantic Ground Truth Construction: We utilize the publicly available CMU Enron Corpus [3] as the structural baseline. For the evaluation of retrieval performance (Recall/Precision), we utilize established relevance judgments from the TREC Legal Track or comparable academic annotation sets (e.g., UC Berkeley Enron Analysis). This ensures that the Ground Truth represents the semantic reality of the documents, independent of the specific vocabulary used in the query.

Comparative Ablation Study: To quantify the added value of the hybrid architecture, we will conduct an ablation study comparing three configurations:

(a) Purely Lexical (Level 1 BM25),
(b) Purely Semantic (Level 2 Vector-only), and
(c) Hybrid Score Fusion (Level 1 + Level 2).

We hypothesize that the hybrid system yields the highest Recall@100, demonstrating superior robustness in scenarios where suspects employ obfuscated language or indirect phrasing that evades purely lexical detection.

References

[1] M. Skipanes, G. Demartini, K. Franke, and A. B. Nissen, "Information analysis in criminal investigations: Methods, challenges, and computational opportunities processing unstructured text," *Policing: A Journal of Policy and Practice*, vol. 19, paaf005, Mar. 2025, ISSN: 1752-4520. DOI: 10.1093/police/paaf005.

[2] OpenSearch Project, *OpenSearch*, version 3.4, https://opensearch.org/ [visited: 2026-03-13], The Linux Foundation, 2025.

[3] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Machine Learning: ECML 2004*, J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 217–226, ISBN: 978-3-540-30115-8. DOI: 10.1007/978-3-540-30115-8_22.

[4] Y. Zhu et al., "Large language models for information retrieval: A survey," *ACM Transactions on Information Systems*, vol. 44, no. 1, pp. 1–54, 2026, ISSN: 1046-8188. DOI: 10.1145/3748304.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[5] L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang, "A survey on employing large language models for text-to-sql tasks," *ACM Computing Surveys*, vol. 58, no. 2, pp. 1–37, 2026, ISSN: 0360-0300. DOI: 10.1145/3737873.

[6] Y. Gao et al., *Retrieval-augmented generation for large language models: A survey*, 2024. DOI: 10.48550/arXiv.2312.10997.

[7] N. F. Liu et al., "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024. DOI: 10.1162/tacl_a_00638.

[8] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo et al., Eds., vol. 35, Curran Associates, Inc., 2022, pp. 24 824–24 837.

[9] Langflow AI, *Langflow*, version 1.6.8, https://github.com/langflow-ai/langflow [visited: 2026-03-13], 2025.

[10] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2019. DOI: 10.48550/arXiv.1908.10084.

[11] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008, ISBN: 9780521865715. DOI: 10.1017/CBO9780511809071.

[12] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3," in *Overview of the Third Text REtrieval Conference (TREC-3)*, NIST, 1995, pp. 109–126.

[13] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020. DOI: 10.1109/TPAMI.2018.2889473.

[14] P. Lewis et al., "Retrieval-Augmented Generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Power-Aware Container Placement Mechanism for CaaS Systems

Abdulelah Alwabel
Department of Computer Sciences
Prince Sattam Bin Abdulaziz University
AlKharj, Saudi Arabia
e-mail: {a.alwabel}@psau.edu.sa

***Abstract*—Containers-as-a-Service (CaaS) platforms are widely adopted due to the lightweight and portable nature of containerized-based applications. However, the growing complexity of container workloads makes it difficult to both achieve efficient resource allocation and reduced energy consumption. To address this challenge, we propose a Power-aware Container Placement (PCP) mechanism, , which maps containers to physical machines (PMs) in cloud data centers with an aim to minimize power consumption while maintaining an acceptable level of performance. The PCP mechanism utilizes the Whale Optimization Algorithm (WOA) to search for energy-aware placements under resource constraints, and then applies an additional best-fit optimization phase to further consolidate and reduce the number of active PMs. The proposed approach is evaluated using a Java-based simulation and is compared against the Extend Directed Container Placement (EDCP) mechanism from the literature. Experimental results show that the proposed mechanism reduces power consumption by approximately 7.4% in comparison to the EDCP mechanism. In addition, it also achieves a about 5% reduction in search time. These results indicate that our mechanism can improve energy efficiency in CaaS systems with minimum negative impact on performance of the systems.**

Keywords-Container; Container Placement; CaaS; PCP; Cloud; Optimization.

I. Introduction

Container Technology is widely employed in Cloud systems as a Container-as-a-Service (CaaS) because they are lightweight and highly portable [1]. However, the rapid growth in the size and complexity of container workloads makes it more challenging to allocate resources effectively while meeting performance requirements and limiting energy consumption. Container placement plays a crucial role in tackling these challenges, as it specifies how containers are assigned to Physical Machines (PMs). Effective placement strategies can improve data-center efficiency by reducing operational costs, lowering power usage, and increasing resource utilization [2].

In real-world deployments, container-based services often consist of many isolated containers, which increases the importance of efficient scheduling and orchestration [3]. As a result, a wide range of placement methods has been proposed, including optimization-based techniques, such as linear programming (e.g., [4]) and heuristic or metaheuristic algorithms (e.g., [5]). Although these approaches can enhance energy efficiency, utilization, or network cost, they may suffer from high computational complexity, added runtime overhead, or limited scalability in large-scale environments. Hybrid optimization approaches are therefore considered a promising alternative, as they can simultaneously balance multiple objectives, such as energy efficiency, load distribution, and quality of service (QoS). This paper introduces a new container placement mechanism that aims to reduce power consumption in CaaS systems while preserving an acceptable performance level compared with existing metaheuristic solutions. The proposed work extends our previous works in [6] and [7] with an aim of further enhancing power efficiency.

The rest of the paper is structured as follows. Section II surveys the related literature. Section III details the proposed approach. Section IV reports the evaluation methodology and discusses the results. Section V concludes the paper.

II. Related Work

The work in [8] presented an energy-aware scheduling model that applies particle swarm optimization. Results showed notable energy savings while preserving an acceptable QoS level.

The authors in [9] proposed an optimization method based on the ant colony optimization algorithm [10] to balance resource utilization, network consumption, and failure events in microservices cloud applications. Their mechanism improved service reliability, load balancing, and network transmission overhead, but it did not account for energy consumption.

A heterogeneous container placement strategy was proposed in [11] to improve the cost efficiency of container orchestration in Kubernetes-based cloud systems. Experiments demonstrated considerable cost reductions compared to default Kubernetes behavior under different workload patterns, although the approach was not evaluated under high service-demand scenarios.

In [12], a locality-aware scheduling model was introduced to boost the performance of containerized cloud services by mitigating resource contention and improving network efficiency. The model employs load-balancing heuristics to enhance application performance; however, performance may degrade sharply as workload size grows [13].

The authors in [14] proposed a scaling mechanism that adapts groups of containers to changing workloads to improve Quality of Experience (QoE). Likewise, [15] focuses on scheduling that prioritizes QoE over QoS by using deep learning to integrate QoE indicators, offering a more faithful representation of user satisfaction for QoE prediction. Their experiments showed an average QoE improvement of about 62% compared with traditional schedulers.

The researchers in [5] developed a whale-optimization-based container placement method [16] to improve resource utilization and reduce power consumption in cloud systems. The

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

results indicated better performance than competing placement mechanisms in heterogeneous test settings. However, the study did not assess potential performance overheads, such as resource overutilization, associated with the method.

The authors in [17] proposed a multi-objective container migration strategy based on a Binary Grey Wolf Optimizer to improve resource utilization and energy efficiency. To reduce migration inefficiencies, they introduced a node coordination matrix model to address resource fragmentation. The evaluation showed that the proposed strategy outperformed existing mechanisms, suggesting practical effectiveness, yet it did not examine scenarios involving very large containers or large numbers of PMs.

Our earlier work in [6] presented a whale-optimization-based container placement mechanism aimed at reducing energy consumption in cloud systems. However, it adversely affected performance metrics such as service violations and search time. This was extended in [7] by introducing a new scoring scheme to better balance performance with power usage. Although performance improved, some power overhead remained. However, both approaches could be further improved by reducing power consumption while maintaining minimal impact on performance, as will be discussed later.

The authors in [18] introduced a container consolidation approach for CaaS clouds using Fractional Pelican Hawks Optimization. The method includes (i) a host-status module to predict PM load and (ii) a consolidation module to manage container migrations. It targets multiple objectives—energy consumption, resource utilization, and cost—leading to notable efficiency gains. Nonetheless, the study did not evaluate scalability for larger, more complex cloud infrastructures.

Gouaouri et al. [19] study power-aware workload scheduling in Kubernetes, arguing that default rule-based scheduling can lead to higher power consumption and inefficient cluster operation at scale. They propose a two-stage, multi-objective framework that uses an ML-based power predictor trained from real node profiling and then performs online scheduling via TOPSIS and NSGA-II. The solution is implemented as a scalable Kubernetes scheduler plugin that can integrate with other plugins and lets users extend objectives based on QoS needs, explicitly optimizing consolidation (active nodes), load balancing, and utilization efficiency together.

The study proposed by [20] address dynamic container placement for serverless edge computing, where network conditions and limited resources make static placement ineffective and cold-start delays can hurt latency-sensitive tasks. They model placement as an online convex optimization problem and develop a strongly adaptive container placement algorithm with a regret guarantee to minimize average request delay using iterative feedback-based refinement.

The authors in [21] focus on container-based microservice scheduling, noting that existing approaches can suffer from poor load balancing, high network overhead, and weaker reliability. They introduce an enhanced multi-objective PSO that uses an indicator for controlled progress plus complementary indicators to better evaluate trade-offs. The reported results indicate that the proposed mechanism reduces load imbalance and network transmission while improving service reliability in heterogeneous cloud environments.

The authors in [22] propose an SLO-aware container orchestration approach for Kubernetes, motivated by the difficulty of meeting response-time SLOs without costly over-provisioning and by the fact that conventional autoscalers rely on low-level CPU/memory metrics.

Algorithm 1 PCP Mechanism

input: c_l, p_l
initialize s_l as a list of n_s random solutions
for i to $itrN$ **do**
  $b \leftarrow findBestSolution(s_l)$
  foreach s in s_l **do** //update the current solution s
    Let s_r be a random solution
    $cf_1 \leftarrow 2 \times a \times rand$
    $cf_2 \leftarrow 2 \times rand$
    Let $prob$ be a random number from 0 to 1
    if $prob < 0.5$ **then**
      $dir \leftarrow |cf_2 \times s_r - s|$
      if $|cf_1| < 1$ **then**
        $s \leftarrow b - cf_1 \times dir$
      else
        $s \leftarrow s_r - cf_1 \times dir$
      end if
    else
      $dir \leftarrow (|b - s|)$
      Let z be a random number from -1 to 1
      $s \leftarrow dir \times e^z \times cos(2\pi z) \times s$
    end if
  end for
end for
$b \leftarrow optimizeSolution(b)$
return: b

Algorithm 2 Optimization Mechanism

input: b
$containerList \leftarrow b.removeContinaerList()$
$pmList \leftarrow b.removePmList()$
sort $containerList$ by CPU requirement ascending
foreach $container$ in $containerList$ **do**
  foreach pm in $pmList$ **do**
    if $pm.isBestFit()$ **then**
      $pm.host(container)$
      $pmList.add(pm)$
    end if
  end for
end for
$b.setPMList(pmList)$
return: b

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

III. Proposed Mechanism

In this section, we propose Power-aware Container Placement (PCP) mechanism that places a list of container to a list PMs in a CaaS system with an aim to reduce power consumption. The PCP extends our previous work presented in [7]. The mechanism employs whale optimization algorithm (WOA) that is proposed by [16]. The WOA is a nature-inspired meta-heuristic optimization method that mimics humpback whales' bubble-net hunting behavior. The algorithm begins with a set of random solutions, then in each iteration the search agents update their positions toward either the best solution so far or a random agent, switching between spiral and circular movement to model bubble-net attacking.

The PCP mechanism is depicted in Algorithm 1. It starts by creating a list of n_s number of random solutions, called s_l. Each solution represents placing a container list c_l to a list of PMs found in p_l in a random way. Then, the mechanism finds the best solution, called b, among a list of solutions using a method called $findBestSolution$. The best solution is defined as the solution which places containers to PMs in a manner that improves performance and reduces power consumption as it is explained in [7]. Each solution s_1 is evaluated against another solution s_2 according to the following:

$$ev(s_1, s_2) = \frac{pw(s_1) - pw(s_2)}{pw(s_2)} + \frac{o(s_1) - o(s_2)}{o(s_2)} \quad (1)$$

where $pw(s)$ is the power consumed by the solution s and is calculated as [23]:

$$pw(s) = \sum_{i=1}^{p} (pm_i^{idle} + (pm_i^m - pm_i^{idle}) \times \frac{pm_i^c}{pm_i^t}) \quad (2)$$

where p refers to the total number of PMs in a CaaS system. pm_i^{idle} denotes the power consumption by pm_i when it is idle. pm_i^m refers to the maximum power consumed by this PM. pm_i^c and pm_i^t denote the current CPU and total CPU of pm_i respectively.

$o(s)$ in (1) denotes the number of PMs in a solution where each PM's utilization level exceeds a certain threshold, called ou. If a utilization level in a PM exceeds a certain threshold, this can lead to a negative impact on the performance of this PM [24].

b is, then, used to update other solutions based on a method called $updateSolution$ according to [5]. The same process is repeated for a number of iterations called $itrN$ as it is explained in [6]. The PCP mechanism repeats this process of updating solutions in s_l until $itrN$.

The last step is to further optimize b one last time using an optimization approach as it is depicted in Algorithm 2. The optimization mechanism employs best fit (BF) approach in order to reduce the number of active PMs in the system. It sorts containers by CPU requirements in an ascending manner. Then, a single PM will host as many containers as it can handle in terms of CPU and memory powers. Lastly, the updated best solution is returned as a result of this optimization phase.

TABLE I. Experiment Configurations

Parameter	Values
Container Specifications:	
Number	2000
CPU Requirements (MIPS)	256, 512, 1024, 2048, 4096
Memory Requirements (MB)	128, 256, 512, 1024, 2048
PM Specifications:	
No of PMs	1000
No of Cores	6, 8, 20, 32, 56, 64
CPU (GHz)	2.2, 2.7, 2.9, 3.2
Memory (GB)	16, 128, 192, 512, 768
pm_{idle} (watt)	15.6, 21.7, 48.6, 53.2, 127
pm_m (watt)	58.9, 82.8, 269, 291, 377, 410
Simulation Settings:	
n_s	5
$itrN$	10
ou	80%

IV. Evaluation

This section discusses the results of PCP mechanism and compares it with EDCP mechanism [7]. The EDCP mechanism is a container placement mechanism based on Meta-heuristic optimization algorithm that places containers with an aim to find an optimum solution in an acceptable time. The mechanisms are evaluated in terms of performance and energy consumption.

A. Experiment Configurations

Table I shows the configurations of the experiments conducted in a Java simulation that was extended based on a work by [6] to evaluate the EDCP and PCP mechanisms. The total number of container is 2000 that submitted to a CaaS of PM machines. The specifications of the PMs are listed in the table. The table shows power consumption of each PM when it is idle and when it is at maximum loading. ou is set to 80% because several studies show that when CPU utilization exceeds this level, the system may experience performance degradation [25].

B. Results

The PCP mechanism outperformed the EDCP mechanism in reducing power consumption as it is depicted in Figure 1. The Figure clearly demonstrates that the gap in power consumption increases when the number of container increases. Overall, the proposed mechanism reduces the power by about 7.4% in compared to the EDCP mechanism.

With respect to search time, PCP also achieves an improvement in by reducing the search time to find an optimum solution by about 5% on average. The EDCP scored about 0.27

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

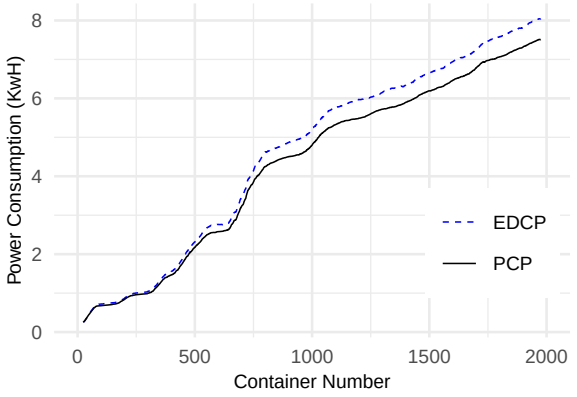


Figure 1. Power Consumption

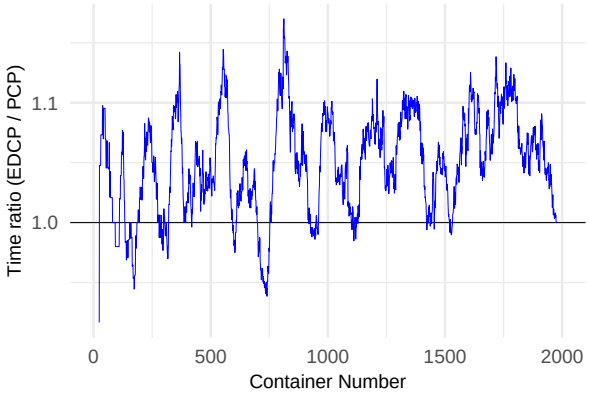


Figure 2. Search Time

seconds on average to find the best solution while the PCP achieved about 0.26 seconds on average. The t-test analysis demonstrates that analysis the PCP achieved a significantly lower runtime than the EDCP (paired t-test: t(2000)=7.63, $p < 0.001$). Figure 2 demonstrates the time ratio between EDCP and PCP which is inconsistent. That means although the PCP mechanism managed to reduce time in general there are some instances that the EDCP mechanism behaved better.

V. Conclusion and Future Work

Container technologies are widely adopted in CaaS environments due to their lightweight virtualization and portability. Container placement can play a key role in tackling several challenges in CaaS including effective resource allocation and limiting energy consumption. Effective container placement solutions can improve efficiency in clouds' infrastructure by reducing operational costs, lowering power usage, and increasing resource utilization.

In this paper, we presented PCP, a novel container placement mechanism that maps containers to PMs in cloud data centers with the objective of minimizing energy consumption. The proposed approach adapts the WOA to search for an energy-aware placement that satisfies resource constraints. Moreover, PCP refines an existing optimization technique from the literature to further reduce power consumption. Experimental results demonstrate that PCP achieves approximately a 7.4% reduction in power consumption compared with EDCP.

For future work, the proposed mechanism may be extended from static placement to dynamic runtime management by supporting container or VM migrations among PMs whenever workload fluctuations or resource imbalance occur. Moreover, the heterogeneity of physical machines can be incorporated more explicitly into the placement logic by accounting for variations in CPU capacity, memory resources, and power profiles. The current evaluation also relies on a simplified linear power model based on CPU utilization; therefore, it would be valuable to investigate whether the effectiveness of the proposed approach is preserved under more sophisticated non-linear or empirical power models. Furthermore, stronger performance guarantees for hosted containers could be introduced by integrating QoS/SLO-aware objectives and constraints, such as latency, response time, and service stability, instead of relying primarily on CPU-utilization threshold control. In addition, PCP can be extended with reliability-aware capabilities to improve service continuity in CaaS environments. Large-scale CaaS infrastructures typically operate a substantial number of PMs, which increases the likelihood of host failures and may consequently degrade SLOs. Incorporating fault-tolerance mechanisms (e.g., replication, proactive migration, or failure-aware placement constraints) could mitigate these disruptions and enhance overall system reliability.

References

[1] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes", *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sep. 2014, ISSN: 2325-6095. DOI: 10.1109/MCC.2014.51.

[2] I. Ahmad, M. G. AlFailakawi, A. AlMutawa, and L. Alsalman, "Container scheduling techniques: A Survey and assessment", *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3934–3947, Jul. 2022, ISSN: 13191578. DOI: 10.1016/j.jksuci.2021.03.002.

[3] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for High Performance Computing Systems: Survey and Prospects", *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2722–2740, Apr. 2023, ISSN: 0098-5589. DOI: 10.1109/TSE.2022.3229221. arXiv: 2212.08717.

[4] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint Container Placement and Task Provisioning in Dynamic Fog Computing", *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 028–10 040, 2019, ISSN: 23274662. DOI: 10.1109/JIOT.2019.2935056.

[5] A. Al-Moalmi, J. Luo, A. Salah, K. Li, and L. Yin, "A whale optimization system for energy-efficient container placement in data centers", *Expert Systems with Applications*, vol. 164, p. 113 719, Feb. 2021, ISSN: 09574174. DOI: 10.1016/j.eswa.2020.113719.

[6] A. Alwabel, "A Novel Container Placement Mechanism Based on Whale Optimization Algorithm for CaaS Clouds", *Electronics*, vol. 12, no. 15, p. 3369, Aug. 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12153369.

[7] A. Alwabel, "An extended container placement mechanism to enhance the efficiency of cloud systems", *PeerJ Computer Science*, vol. 11, e3348, Nov. 2025, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.3348.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[8] T. Shi, H. Ma, and G. Chen, "Energy-Aware Container Consolidation Based on PSO in Cloud Data Centers", in *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, Jul. 2018, pp. 1–8, ISBN: 978-1-5090-6017-7. DOI: 10.1109/CEC.2018.8477708.

[9] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud", *IEEE Access*, vol. 7, pp. 83 088–83 100, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2924414.

[10] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization", *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, Nov. 2006, ISSN: 1556-603X. DOI: 10.1109/MCI.2006.329691.

[11] Z. Zhong and R. Buyya, "A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources", *ACM Transactions on Internet Technology*, vol. 20, no. 2, pp. 1–24, May 2020, ISSN: 1533-5399. DOI: 10.1145/3378447.

[12] D. Zhao, M. Mohamed, and H. Ludwig, "Locality-Aware Scheduling for Containers in Cloud Computing", *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 635–646, Apr. 2020, ISSN: 2168-7161. DOI: 10.1109/TCC.2018.2794344.

[13] S. Deng et al., "Cloud-Native Computing: A Survey From the Perspective of Services", *Proceedings of the IEEE*, vol. 112, no. 1, pp. 12–46, Jan. 2024, ISSN: 0018-9219. DOI: 10.1109/JPROC.2024.3353855.

[14] G. Santos, H. Paulino, and T. Vardasca, "QoE-aware autoscaling of heterogeneous containerized services (and its application to health services)", in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, New York, NY, USA: ACM, Mar. 2020, pp. 242–249, ISBN: 9781450368667. DOI: 10.1145/3341105.3373915.

[15] M. Carvalho and D. F. Macedo, "Container Scheduling in Co-Located Environments Using QoE Awareness", *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3247–3260, Sep. 2023, ISSN: 1932-4537. DOI: 10.1109/TNSM.2023.3244090.

[16] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm", *Advances in Engineering Software*, vol. 95, pp. 51–67, May 2016, ISSN: 09659978. DOI: 10.1016/j.advengsoft.2016.01.008.

[17] C. L. Chuqiao Lin et al., "Container Migration Strategy Based on Multi-objective Optimization for Edge-Cloud Coordination enabled Smart Grids", *Journal of Computers*, vol. 34, no. 6, pp. 047–062, Dec. 2023, ISSN: 19911599. DOI: 10.53106/199115992023123406004.

[18] M. K. Patra, B. Sahoo, and A. K. Turuk, "FPHO: Fractional Pelican Hawks optimization based container consolidation in CaaS cloud", *Concurrency and Computation: Practice and Experience*, vol. 36, no. 12, e8052, May 2024, ISSN: 1532-0626. DOI: 10.1002/cpe.8052.

[19] M. D. E. Gouaouri, S. Ouahouah, M. Bagaa, M. A. Ouameur, and A. Ksentini, "A multi-objective framework for power-aware scheduling in kubernetes", *IEEE Transactions on Network and Service Management*, vol. 23, pp. 28–46, 2026, ISSN: 1932-4537. DOI: 10.1109/TNSM.2025.3630045.

[20] S. Cao and S. Wang, "Dynamic container placement at serverless edge: An online convex optimization method", in *2025 IEEE 34th Wireless and Optical Communications Conference (WOCC)*, IEEE, May 2025, pp. 66–70, ISBN: 979-8-3315-3928-3. DOI: 10.1109/WOCC63563.2025.11082211.

[21] M. Douiri, I. B. Mansour, and M. Tagina, "Optimizing container-based microservice scheduling with parallel particle swarm based on diversity indicators", in *2025 IEEE 37th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, Nov. 2025, pp. 81–88, ISBN: 979-8-3315-4919-0. DOI: 10.1109/ICTAI66417.2025.00019.

[22] A. Marchese and O. Tomarchio, "Slo-aware container orchestration on kubernetes clusters", in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, IEEE, Jul. 2025, pp. 318–327, ISBN: 979-8-3315-5557-3. DOI: 10.1109/CLOUD67622.2025.00040.

[23] A. A. Khan et al., "An energy and performance aware consolidation technique for containerized datacenters", *IEEE Transactions on Cloud Computing*, 2019, ISSN: 2168-7161. DOI: 10.1109/tcc.2019.2920914.

[24] I. Çağlar and D. T. Altılar, "Look-ahead energy efficient VM allocation approach for data centers", *Journal of Cloud Computing*, vol. 11, no. 1, p. 11, Dec. 2022, ISSN: 2192-113X. DOI: 10.1186/s13677-022-00281-x.

[25] B. Gregg, *Systems Performance: Enterprise and the Cloud*, 1st. Pearson Education, Inc., 2014.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

From Intent to Deploy: Validator-Centric Multi-Agent Orchestration for Reliable Infrastructure-as-Code

Rana Nameer Hussain Khan, Dawood Wasif, Jin-Hee Cho, Ali R. Butt
Department of Computer Science
Virginia Tech
Blacksburg, VA, USA
e-mail: rnameerkhan@vt.edu, dawoodwasif@vt.edu, jicho@vt.edu, butta@vt.edu

***Abstract*—The increasing complexity of cloud-native infrastructure has made Infrastructure-as-Code (IaC) essential for reproducible and scalable deployments. While Large Language Models (LLMs) have shown promise in generating IaC snippets from natural language prompts, their monolithic, single-pass generation approach often results in syntactic errors, policy violations, and unscalable designs. In this paper, we propose MACOG (Multi-Agent Code-Orchestrated Generation), a novel multi-agent LLM-based architecture for IaC generation that decomposes the task into modular subtasks handled by specialized agents: Architect, Provider Harmonizer, Engineer, Reviewer, Security Prover, Cost and Capacity Planner, DevOps, and Memory Curator. The agents interact via a shared-blackboard, finite-state orchestrator layer, and collectively produce Terraform configurations that are not only syntactically valid but also policy-compliant, semantically coherent. To ensure infrastructure correctness and governance, we incorporate Terraform Plan for execution validation and Open Policy Agent (OPA) for customizable policy enforcement. We evaluate MACOG using the IaC-Eval benchmark, where MACOG is the top enhancement across models, e.g., GPT-5 improves from 54.90 (RAG) to 74.02 and Gemini-2.5 Pro from 43.56 to 60.13, with concurrent gains on BLEU, CodeBERTScore, and an LLM-judge metric. Ablations show constrained decoding and deploy feedback are critical. Removing them drops IaC-Eval to 64.89 and 56.93, respectively.**

Keywords-Infrastructure as Code; multi-agent systems; Large Language Models; program synthesis; policy as code.

I. INTRODUCTION

Cloud platforms expose a rapidly expanding surface of services, configuration parameters, and compliance regimes, making Infrastructure-as-Code (IaC) essential for reproducible deployment and operations [1]. Tools such as Terraform, Pulumi, and CloudFormation encode desired cloud state as declarative programs that must satisfy provider schemas, remain consistent across interdependent resources, and obey organizational constraints on security, cost, and data residency. Large language Models (LLMs) [2] promise to translate natural-language intent into IaC, yet practical correctness remains difficult. Deployments fail on version-sensitive schema details, fragile cross-resource references, and dependency structures that surface only during `plan/apply`. In production, IaC must also enforce least privilege, encryption, region pinning, redundancy targets, and budget limits. IaC synthesis is therefore constrained program construction validated by multiple independent authorities: schema checks, policy engines, cost estimators, and real toolchains.

These challenges are amplified by cloud evolution and modular reuse: provider APIs deprecate fields, migrate services across regions, and shift defaults, while modules are composed across teams and time with mismatched interfaces. IaC is inherently graph-structured, so near-miss generations often fail due to small but decisive mismatches (attributes, module wiring, region capabilities) that surface only under heterogeneous validators: security policies (OPA/Rego and scanners, such as Checkov/Regula), cost estimators, and real `plan/apply` toolchains. Prior work improves reliability through prompting, retrieval-augmented generation [0], and iterative refinement, but these approaches remain brittle under schema drift, multi-module dependencies, and unstable repair loops, and they rarely maintain a typed view of the infrastructure graph. Reliable assistants must therefore place validators at the center of the workflow rather than treat them as optional feedback.

We introduce *Multi-Agent Code-Orchestrated Generation* (MACOG), a validator-centric framework for generating deployable and auditable Terraform from natural-language intent without any model fine-tuning. MACOG (i) represents intent as a typed Infrastructure Intermediate Representation (I-IR) capturing resources, dependencies, regions, and required effects; (ii) compiles I-IR into Terraform using deterministic resource skeletons plus grammar- and schema-constrained decoding that stays within HCL [3] and provider field sets, enforced with a round-trip structural check; and (iii) closes the loop with tool-grounded validation and repair, combining `terraform validate`, policy enforcement (OPA/Rego with complementary scanners), deterministic cost estimation, and sandboxed `plan/apply` to produce structured counterexamples that drive minimal plan- or code-level edits. Agents coordinate over a versioned blackboard and reuse verified typed motifs rather than raw snippets to mitigate version drift. The result is a Terraform program paired with an evidence bundle that supports offline verification and audit; this validator-centric design also implies overhead and depends on the fidelity and availability of the underlying toolchains and sandboxes.

The remainder of this paper is organized as follows. Section II (§II) reviews related work on IaC generation, multi-agent code synthesis, and validator-guided repair. Section III (§III) presents MACOG, including the I-IR representation, blackboard orchestration, constrained compilation, and counterexample-driven repair. Section IV (§IV) describes the experimental setup, benchmarks, baselines, and metrics. Section V (§V) discusses

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Copyright (c) IARIA, 2026. ISBN: 978-1-68558-373-6

results and key trade-offs. Section VI (§VI) concludes and outlines future work.

II. Related Work

This section reviews prior work on IaC generation, multi-agent code synthesis, and validator-guided repair, which together motivate the design of MACOG.

A. LLMs for IaC and Configuration Synthesis

LLM-based IaC synthesis for Terraform [4], CloudFormation [5], and Ansible [6] maps natural-language intent to deployable artifacts under strict schemas and cross-resource references. Evidence from IaC-Eval shows one-shot prompting is brittle: even strong models omit required fields, misuse identifiers, and violate provider constraints without tool feedback [7]. Configuration-quality research further documents "smells" that degrade maintainability and security beyond syntax [9]. Security analyses catalog recurring IaC weaknesses (hard-coded secrets, overly permissive policies), motivating policy-aware validation [10], and large-scale measurements of Terraform practice reveal persistent gaps in adoption and enforcement [11]. Together, these results motivate systems that integrate schema awareness with policy compliance and runtime validation rather than relying only on prompting or retrieval.

B. Multi-Agent and Tool-Augmented Code Generation

A complementary line structures code generation as collaboration among specialized agents and external tools. *ChatDev* shows that role specialization, shared memory, and critique improve end-to-end correctness [0], while *self-collaboration* reports gains from plan–code–test role cycling within one model [12]. For software evolution, *MAGIS* coordinates planning and QA to resolve GitHub issues more reliably than single-agent prompting [13], and *RepairAgent* couples LLMs with a finite-state tool controller for diagnosis, patching, and validation [14]. These systems support orchestrations that route typed artifacts through schema, policy, and runtime validators while preserving already-correct structure.

C. Constrained Decoding and Validator-Guided Repair

Constrained decoding reduces invalid structured outputs by restricting generation to grammar- or schema-admissible tokens. *PICARD* enforces incremental parsing constraints for formal languages such as SQL [15]; *SynCode* applies DFA-style masking for CFGs to guarantee syntactic validity efficiently [16]; and *Grammar-Aligned Decoding* (ASAp) aligns sampling with the constrained conditional distribution to mitigate bias from hard constraints [17]. On the repair side, CEGIS uses counterexamples from verification to drive targeted refinements [18], and recent LLM repair work frames refinement as guided search over failures and partial successes [19]. For IaC, combining grammar/schema-constrained realization with validator-guided repair (static checks, policy engines, and sandboxed `plan/apply`) prevents invalid syntax upfront and turns machine-readable counterexamples into minimal, localized patches.

III. Methodology

This section describes a validator-centric pipeline that turns natural-language intent into deployable, secure, and cost-aware Terraform. The system is organized as a blackboard workflow with role-specialized agents operating over a typed Infrastructure Intermediate Representation (I-IR). It compiles I-IR into Terraform using grammar- and schema-constrained decoding, then iteratively repairs failures using structured counterexamples produced by authoritative external validators. The workflow runs entirely in instruction-following mode with no model fine-tuning, and it enforces correctness through deterministic tooling, explicit typing, and evidence-driven edits.

A. Problem Setup

Input consists of a natural-language intent x and a constraint set C that specifies operational requirements, such as budget, residency, encryption, and availability. Output is a Terraform program T together with an evidence bundle Π that supports independent verification.

a) Typed plan representation (I-IR): We represent the intended infrastructure as a typed resource graph $P = \langle V, E, S \rangle$:

- V are resource nodes with provider-specific types and typed fields.
- E are dependency and connectivity edges that make ordering and wiring explicit.
- S are specifications and effects, including security and compliance obligations.

I-IR makes the infrastructure graph first-class: resources, edges, regions, and obligations are explicit and mechanically checkable.

b) Validation contracts: A candidate Terraform program must satisfy four contracts:

- **Schema validity:** provider schemas and references are correct (`terraform validate` and schema matchers).
- **Policy compliance:** organization rules hold (OPA/Rego plus complementary scanners).
- **Cost feasibility:** cost is within the stated budget using pinned catalogs and line-item summaries.
- **Deployability:** `init/plan/apply` succeeds in controlled sandboxes.

The system treats validator outputs as ground truth and uses them to drive precise repairs.

B. System Overview

MACOG uses a blackboard architecture: each agent reads and writes structured artifacts (typed plans, diffs, diagnostics, logs) and the orchestrator advances a fixed state machine. This design prevents uncontrolled rewrites, preserves correct structure across iterations, and makes every change traceable. As shown in Figure 1, MACOG coordinates agents over a shared blackboard and uses validator feedback.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

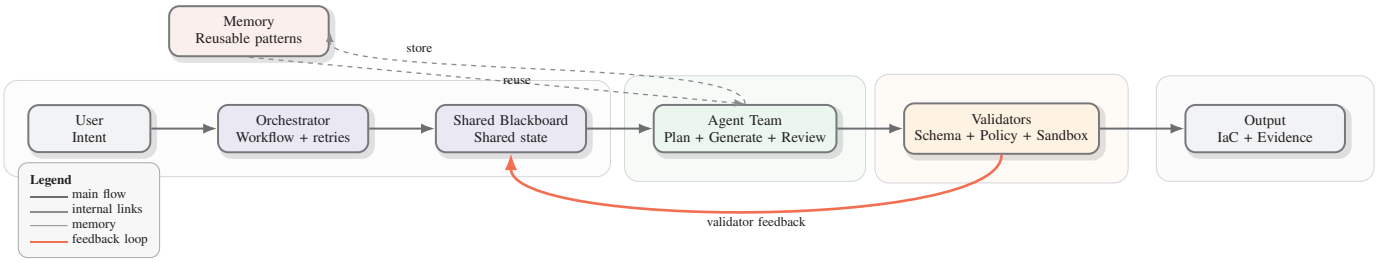


Figure 1. High-level MACOG architecture.

a) Roles and responsibilities:

- **Architect:** converts (x, C) into an initial I-IR plan P_0 and a set of invariants $\mathcal{I}$ that explicitly capture obligations such as encryption, residency, and exposure bounds.
- **Provider Harmonizer:** resolves provider versions, regions, and required fields, producing a fully concrete plan P_1 consistent with pinned schemas.
- **Engineer:** compiles P_1 into Terraform using constrained decoding that enforces HCL grammar and provider field admissibility.
- **Reviewer:** runs static checks (`terraform validate`, linters, module interface checks) and produces structured, localized diagnostics.
- **Security Prover:** executes OPA/Rego and complementary scanners and returns policy traces and counterexample witnesses.
- **Cost & Capacity Planner:** produces deterministic cost sheets from pinned catalogs and flags SKU, quota, and region feasibility issues.
- **DevOps:** runs `init/plan/apply` in sandboxes (LocalStack for fast feedback, ephemeral accounts for realism) and returns runtime error objects and logs.
- **Memory Curator:** stores verified motifs as typed I-IR fragments with schema/version metadata and serves them during planning and compilation.

C. I-IR: Typed Graph with Explicit Obligations

Each resource node carries typed fields and explicit effects:

- **Fields** are typed by pinned provider schemas, so required attributes and admissible values are enforced at the plan level.
- **Effects** encode obligations such as encryption-at-rest and least-privilege, which are later discharged by policy validators.
- **Edges** encode ordering and connectivity, which eliminates hidden dependencies and stabilizes multi-module generation.

This representation eliminates ambiguity: the system always operates on a concrete, checkable infrastructure graph.

D. Constrained Compilation with Round-Trip Checks

Compilation is intentionally strict and proceeds in two stages:

a) Stage 1: Structural skeleton: A structural compiler maps P_1 to a Terraform skeleton $\widetilde{T}$:

- required blocks are emitted first,
- module boundaries, variables, and outputs are constructed deterministically,
- references are filled from a symbol table derived from node identifiers.

b) Stage 2: Constrained decoding: The Engineer completes remaining fields using constrained decoding:

- HCL grammar constraints prevent syntactically invalid output,
- provider-schema constraints prevent illegal fields and incompatible values,
- reference constraints enforce correct attribute wiring and scope.

c) Round-trip equivalence: Before any expensive checks, the system parses the generated Terraform back into a plan and enforces structural equivalence with P_1 modulo harmless normalization (field ordering and renaming). This guarantees that compilation preserves intent and prevents drift across iterations.

E. Validator-Grounded Repair

After compilation, MACOG runs the full validator suite and collects structured counterexamples:

- **Schema counterexamples:** missing required attributes, type mismatches, broken references.
- **Policy counterexamples:** rule identifiers, failed predicates, and witnesses from OPA/Rego and cross-check scanners.
- **Cost counterexamples:** line items exceeding budget, region pricing mismatches, capacity flags.
- **Runtime counterexamples:** `plan/apply` failures including dependency cycles, unsupported SKUs, quota errors, and missing identifiers.

a) Deterministic error-to-edit mapping: Counterexamples are mapped to minimal edits using a fixed set of operators:

- **Plan-level edits** modify I-IR when failures are structural (region, topology, connectivity, required effects).
- **Code-level edits** modify Terraform when failures are local (single field, reference, block attribute).

Edits are applied with a strict priority order: structural fixes first, then localized patching. This produces fast convergence and prevents late-stage thrashing.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE I. AVERAGE IAC-EVAL TASK SUCCESS (%, HIGHER IS BETTER) ACROSS MODELS UNDER FIVE ENHANCEMENT STRATEGIES.

Rank	Name	Few-shot	CoT	Multi-turn	RAG	MACOG
1	GPT-5	12.53	10.19	35.83	54.90	74.02
2	Gemini-2.5 Pro	12.18	10.49	36.81	43.56	60.13
3	GPT-4	10.64	9.31	31.12	36.70	43.20
4	GPT-3.5-turbo	0.80	1.60	11.44	21.81	25.40
5	Gemini 2.0 Flash	3.33	1.80	4.93	10.32	17.85
6	Magicoder-S-CL-7B	2.93	0.53	12.50	12.77	16.95
7	WizardCoder-33B-V1.1	1.60	1.06	9.04	11.70	15.80
8	CodeLlama Instruct (34B)	3.19	3.19	2.13	6.12	10.45
9	CodeLlama Instruct (7B)	2.39	3.72	0.53	5.59	9.70
10	CodeLlama Instruct (13B)	1.06	1.86	1.06	3.46	6.40

b) Routing objective: The orchestrator uses a single compact score to prioritize which failures to address first:

$$J(T,C) = \lambda_s\,(1 - v_{\text{schema}}) + \lambda_p\,(1 - v_{\text{policy}}) + \lambda_d\,(1 - v_{\text{deploy}}) + \lambda_c\,\max(0, \widehat{\text{cost}}(T) - B), \quad (1)$$

where $v_{\text{schema}}, v_{\text{policy}}, v_{\text{deploy}} \in \{0,1\}$ are validator pass indicators, $\widehat{\text{cost}}(T)$ is the deterministic cost estimate, and B is the budget. This score is used only for deterministic control flow and never as a learned loss.

F. Blackboard Orchestration and Reproducibility

All artifacts are written to a typed blackboard: I-IR versions, Terraform candidates, diffs, validator traces, cost sheets, and deploy logs. Every entry is stamped with provider schema versions, toolchain digests, and content hashes.

a) Finite-state orchestration: The orchestrator advances a fixed state machine with the following states:

$$\mathsf{S} \in \left\{ \begin{array}{l} \texttt{plan, harmonize, compile, review,} \\ \texttt{prove, price, deploy, repair, done} \end{array} \right\}.$$

Transitions are guarded by explicit contracts, expressed as preconditions over the blackboard artifacts (e.g., `schema-valid`, `policy-valid`, `budget-valid`, `deploy-valid`) in the sense of Design by Contract [0]. This makes behavior predictable, prevents uncontrolled rewrites, and keeps failures fully diagnosable from the recorded traces.

G. Tooling and Proof-Carrying Output

MACOG returns (T, Π):

- T is the final Terraform program.
- Π is a self-contained evidence bundle with policy traces and witnesses, cost sheets with catalog versions, schema snapshots, static validation logs, deploy logs, round-trip equivalence records, and the complete repair trajectory.

This output is audit-ready: the same validators can replay Π offline and confirm compliance before production execution.

IV. EXPERIMENTS

We evaluate MACOG on IaC-Eval [7], comparing it against four commonly used enhancement strategies (Few-shot, CoT, Multi-turn repair, and RAG) across a broad set of code-capable LLMs. We report a cross-model summary on harness-verified task success (IaC-Eval), then provide multi-metric profiles for two frontier models (GPT-5 and Gemini-2.5 Pro), and finally isolate MACOG's key components via ablations. All runs use the same task prompts, the same retry budget, and no fine-tuning.

TABLE II. GPT-5 UNDER FIVE ENHANCEMENT STRATEGIES ACROSS FOUR METRICS (0–100, HIGHER IS BETTER).

Strategy	BLEU	CodeBERTScore	LLM-judge	IaC-Eval
Few-shot	5.68	72.41	68.22	12.53
CoT	3.37	70.85	60.31	10.19
Multi-turn	5.54	71.08	62.17	35.83
RAG	10.71	76.43	69.72	54.90
MACOG	**11.86**	**80.54**	**94.10**	**74.02**

TABLE III. GEMINI-2.5 PRO UNDER FIVE ENHANCEMENT STRATEGIES ACROSS FOUR METRICS (0–100, HIGHER IS BETTER).

Strategy	BLEU	CodeBERTScore	LLM-judge	IaC-Eval
Few-shot	5.12	65.08	57.41	12.18
CoT	4.94	61.77	56.20	10.49
Multi-turn	8.87	66.95	58.03	36.81
RAG	9.73	69.92	64.15	43.56
MACOG	**10.09**	**71.84**	**87.52**	**60.13**

A. Setup and Protocol

a) Benchmark: IaC-Eval [7] provides natural-language infrastructure intents with verification procedures designed for Terraform-based provisioning. Tasks span networking, compute, storage, IAM, and managed services, and require globally consistent cross-resource references and provider-valid schemas. A task is counted as solved only if it passes the benchmark harness checks.

b) Models: We evaluate closed and open models representative of modern IaC generation: GPT-5, GPT-4, GPT-3.5-turbo [20]; Gemini-2.5 Pro and Gemini 2.0 Flash [21]; and open-weight code models (Magicoder-S-CL-7B, WizardCoder-33B, CodeLlama Instruct 7B/13B/34B). The same strategy logic is applied across models to attribute gains to the strategy rather than model-specific prompt tuning.

c) Strategies: We compare five increasingly structured approaches: **(i) Few-shot** single-turn exemplars, no tools; **(ii) CoT** single-turn with explicit reasoning before code, no tools; **(iii) Multi-turn** bounded retries with validator feedback paraphrased in natural language; **(iv) RAG** retrieval

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

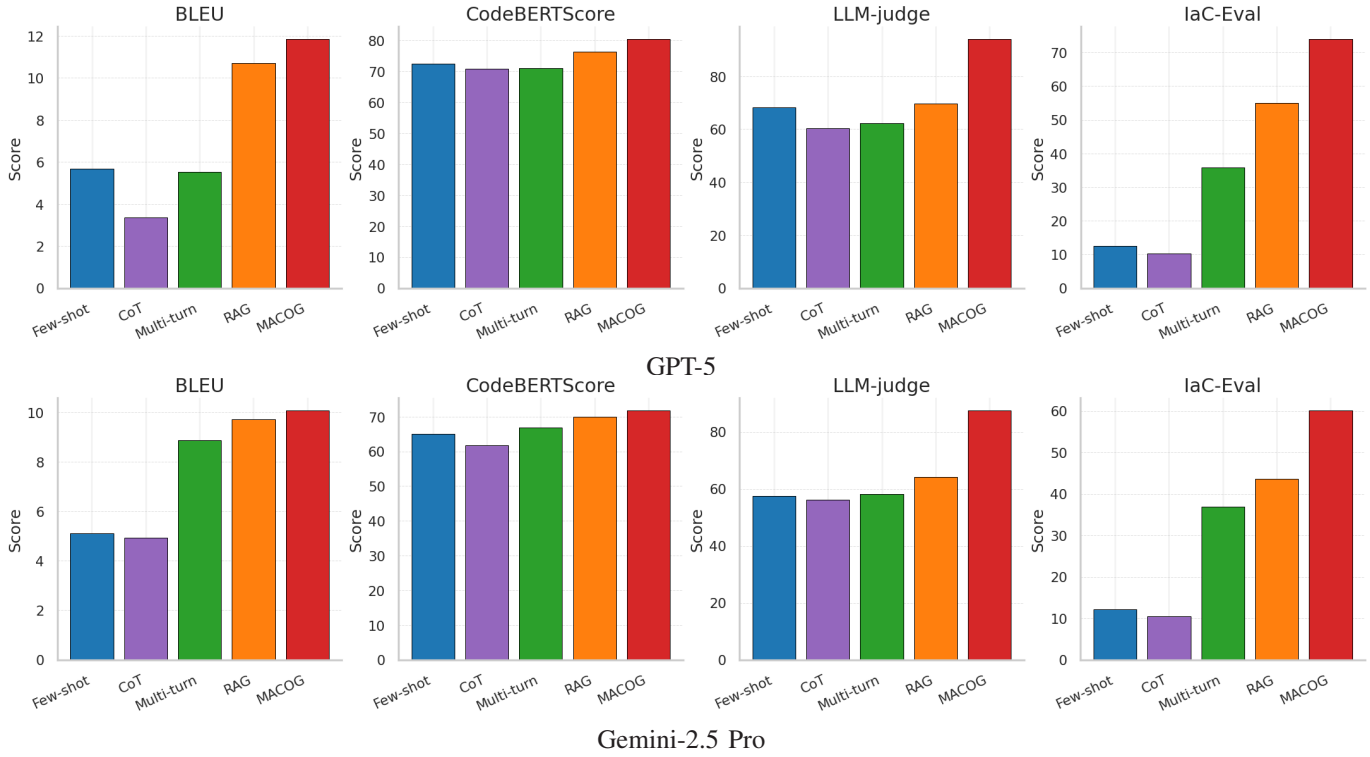


Figure 2. Metric profiles across enhancement strategies (BLEU, CodeBERTScore, LLM-judge, IaC-Eval).

TABLE IV. ABLATION STUDY OF MACOG COMPONENTS ON GPT-5 (0–100, HIGHER IS BETTER).

Variant	BLEU	CodeBERT	Judge	IaC-Eval
Full MACOG	**11.86**	**80.54**	**94.10**	**74.02**
– Harmonizer	10.98	78.92	92.48	70.37
– Engineer (no constr. dec.)	8.61	73.15	89.74	64.89
– Reviewer	10.27	76.04	86.11	66.72
– Security Prover	10.81	77.53	90.03	61.45
– Cost & Capacity	11.22	79.38	92.01	71.08
– DevOps (no sandbox)	9.47	74.82	88.57	56.93
– Memory Curator	10.95	79.06	91.34	72.17

of similar solved tasks as short sanitized hints, no hard constraints; **(v) MACOG** multi-agent orchestration over typed I-IR, grammar/schema-constrained realization, round-trip structural checks, and validator-driven counterexample repairs.

d) Metrics: We report BLEU, CodeBERTScore–F1, LLM-judge, and IaC-Eval using the official IaC-Eval implementations in the code repository [7].

e) Toolchain and validators: All runs use a pinned Terraform toolchain for static validation, provider schema snapshots for required-field/type checks, OPA/Rego policies with a curated ruleset plus a second scanner as a cross-check, and a deployment sandbox (LocalStack and ephemeral accounts) where permitted. Validator outputs are stored as structured JSON traces. Only MACOG consumes structured traces programmatically; Multi-turn receives a natural-language summary; Few-shot/CoT/RAG receive no validator traces.

f) Artifacts and reproducibility: We use the public IaC-Eval benchmark [7], available at https://huggingface.co/datasets/autoiac-project/iac-eval. All code and evaluation scripts are archived on Zenodo at https://zenodo.org/records/17117489.

B. Results

a) Frontier model profiles: Figure 2 and Table II–Table III show that MACOG improves not only IaC-Eval success but also judged adequacy and semantic similarity. For GPT-5, MACOG yields simultaneous gains in BLEU, CodeBERTScore, LLM-judge, and IaC-Eval, with the largest jump in LLM-judge and IaC-Eval, indicating that validator-grounded repairs systematically convert near-miss configurations into deployable and policy-consistent programs. Gemini-2.5 Pro shows the same pattern with a strong uplift in LLM-judge and IaC-Eval, reflecting improved adherence to security and correctness constraints under the same prompt budget.

b) Ablation study: Table IV shows that the main gains come from runtime grounding, policy grounding, and constrained decoding: removing the DevOps sandbox yields the largest IaC-Eval drop ($74.02 \rightarrow 56.93$), followed by removing the Security Prover ($74.02 \rightarrow 61.45$) and disabling constrained decoding ($74.02 \rightarrow 64.89$). The remaining modules provide smaller but consistent gains by aligning plans to schemas early and catching interface errors cheaply.

V. DISCUSSION

The experiments show that IaC generation improves most when correctness is defined and enforced by validators rather

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

than prompt-only fluency: MACOG achieves the highest IaC-Eval success across models (Table I) while also improving adequacy (LLM-judge) and semantic similarity (CodeBERTScore) for GPT-5 and Gemini-2.5 Pro (Table II, Table III). These gains follow from the design: typed planning and grammar/schema-constrained realization prevent many structural and field-level errors by construction, and structured counterexamples from schema, policy, cost, and deployment validators drive localized repairs instead of broad rewrites, which stabilizes convergence under real toolchains. Ablations support this mechanism: removing the DevOps sandbox yields the largest drop in success (Table IV), indicating runtime-grounded feedback is decisive for last-mile failures that static checks miss, while removing the Security Prover and constrained decoding produces the next largest degradations, highlighting the value of policy witnesses and admissible decoding. The remaining modules contribute steady improvements by aligning plans with provider realities and catching cheap interface defects early, and the same principles extend beyond Terraform by swapping the compiler backend and validator suite. In terms of overhead, MACOG trades single-shot latency for higher acceptance by paying additional runtime for multi-agent coordination and repeated validator execution, with `plan/apply` typically dominating end-to-end time.

VI. Conclusion and Future Work

This paper presented MACOG, a multi-agent, validator-centric pipeline that synthesizes deployable and compliant Infrastructure-as-Code from natural-language intent without fine-tuning. MACOG uses a typed infrastructure intermediate representation to preserve global structure, grammar- and schema-constrained compilation with round-trip checks to prevent drift, and counterexample-guided repair grounded in static validation, policy enforcement, cost estimation, and sandboxed `plan/apply`. Across ten models and five enhancement strategies, MACOG achieves the highest harness-verified task success on IaC-Eval and improves adequacy and semantic similarity for frontier models (Table I, Table II, Table III); ablations confirm that constrained realization, policy grounding, and runtime grounding account for most gains (Table IV).

Future work will reduce latency via parallel validation, caching, and incremental re-checks, and improve robustness to provider evolution through automated schema/catalog refresh and drift-triggered re-harmonization. Additional directions include richer policies (e.g., residency and supply-chain) with stronger witnesses, multi-cloud backends via swappable compiler/validator suites, and replayable evidence bundles for offline audit and CI/CD gating.

References

[1] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research", *Information and Software Technology*, vol. 108, pp. 65–77, 2019.

[2] Y. Chang et al., "A survey on evaluation of large language models", *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[0] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks", *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[3] P. Riti and D. Flynn, *Beginning HCL Programming*. Springer, 2021.

[4] K. Shirinkin, *Getting Started with Terraform*. Packt Publishing Ltd, 2017.

[5] K. Tovmasyan, *Mastering AWS CloudFormation: Plan, develop, and deploy your cloud infrastructure effectively using AWS CloudFormation*. Packt Publishing Ltd, 2020.

[6] J. McAllister, *Implementing DevOps with Ansible 2*. Packt Publishing Ltd, 2017.

[7] P. T. Kon et al., "Iac-eval: A code generation benchmark for cloud infrastructure-as-code programs", *Advances in Neural Information Processing Systems*, vol. 37, pp. 134 488–134 506, 2024.

[9] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code", in *2018 11Th international conference on the quality of information and communications technology (QUATIC)*, IEEE, 2018, pp. 220–228.

[10] L. C. Opara, O. N. Akatakpo, I. C. Ironuru, K. Anyaene, and B. O. Enobakhare, "Chaos engineering 2.0: A review of ai-driven, policy-guided resilience for multi-cloud systems", *Journal of Computer, Software, and Program*, vol. 2, no. 2, pp. 10–24, 2025.

[11] A. Verdet, M. Hamdaqa, L. D. Silva, and F. Khomh, "Exploring security practices in infrastructure as code: An empirical study", *arXiv preprint arXiv:2308.03952*, 2023. arXiv: 2308.03952 [cs.CR].

[0] C. Qian et al., "Chatdev: Communicative agents for software development", in *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, 2024, pp. 15 174–15 186.

[12] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via chatgpt", *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, pp. 1–38, 2024.

[13] W. Tao et al., "Magis: Llm-based multi-agent framework for github issue resolution", *Advances in Neural Information Processing Systems*, vol. 37, pp. 51 963–51 993, 2024.

[14] I. Bouzenia, P. Devanbu, and M. Pradel, "Repairagent: An autonomous, llm-based agent for program repair", *arXiv preprint arXiv:2403.17134*, 2024.

[15] T. Scholak, N. Schucher, and D. Bahdanau, "Picard: Parsing incrementally for constrained auto-regressive decoding from language models", *arXiv preprint arXiv:2109.05093*, 2021.

[16] S. Ugare, T. Suresh, H. Kang, S. Misailovic, and G. Singh, "Syncode: Llm generation with grammar augmentation", *Transactions on Machine Learning Research*, 2024.

[17] K. Park, J. Wang, T. Berg-Kirkpatrick, N. Polikarpova, and L. D'Antoni, "Grammar-aligned decoding", *Advances in Neural Information Processing Systems*, vol. 37, pp. 24 547–24 568, 2024.

[18] A. Solar-Lezama, "The sketching approach to program synthesis", in *Asian symposium on programming languages and systems*, Springer, 2009, pp. 4–13.

[19] H. Tang et al., "Code repair with llms gives an exploration-exploitation tradeoff", *Advances in Neural Information Processing Systems*, vol. 37, pp. 117 954–117 996, 2024.

[0] B. Meyer, "Applying "design by contract"", *Computer*, vol. 25, no. 10, pp. 40–51, 1992. DOI: 10.1109/2.161279.

[20] OpenAI, "Introducing `chatgpt`", Nov. 2022, Accessed: Nov. 15, 2025. [Online]. Available: https://openai.com/index/chatgpt/.

[21] G. Team et al., "Gemini: A family of highly capable multimodal models", *arXiv preprint arXiv:2312.11805*, 2023.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org