femington

Int. J. Intell. Syst. Data Sci. (2026) 1(3):1–14
Article ID: IJISDS-2026-130

Pi-Scale Mediation: An Empirical Study on Tangible Container Clusters as Pedagogical Catalysts in Distributed Systems Education

Sheshukumar Vangala
sheshuk80@gmail.com
Independent Researcher

Abstract

This paper presents an empirical investigation into the efficacy of tangible, small-scale cloud computing clusters as pedagogical mediators for bridging the conceptual gap in distributed systems education. While cloud computing and microservices have reshaped the software industry, academic curricula often lag due to the abstract nature and high cost of real-world infrastructure. We designed, built, and evaluated KubeCloud, a low-cost, portable cluster comprising four Raspberry Pi nodes running Docker and Kubernetes. This physical learning object served as the centrepiece of a seven-week, problem-based learning activity for engineering students. Through iterative experimentation including cable-pulling fault injection, application-level resilience testing with circuit breakers, and infrastructure-level replication studies, we quantified the cluster's impact on student comprehension. Evaluations based on SOLO and Bloom's taxonomies demonstrated a measurable progression from pre-structural to relational understanding. Findings indicate that tangible clusters significantly enhance the visibility of cloud abstractions, improve grasp of resilience concepts, and align engineering students' sensory-visual learning preferences with complex distributed systems paradigms. The study concludes that such mediated, hands-on environments are not only viable but essential for preparing graduates with the practical, failure-embracing mindset required by modern IT landscapes.

Keywords

• Cloud Computing Education • Raspberry Pi • Kubernetes • Docker • Tangible Learning Objects • Distributed Systems • Resilience

1. INTRODUCTION

1.1 The Gap Between Industry and Academia

Heterogeneous infrastructure includes variations in compute (CPU vs GPU), network bandwidth (Mbps to Gbps), and latency across distributed environments.

Cloud computing has fundamentally transformed how software is built and deployed. Companies large and small have migrated from monolithic applications to microservice architectures running on container orchestration platforms[1, 2]. Netflix runs thousands of microservices across multiple regions. Google deploys billions of containers weekly. Amazon Web Services provides infrastructure that powers a significant portion of the internet. This shift has created enormous demand for graduates who understand distributed systems, containers, and cloud-native architectures [3].

Yet university curricula have struggled to keep pace. Traditional computer science education still emphasizes single-machine programming and monolithic application development. Students learn about threads and processes, but rarely about service discovery. They study algorithms but not circuit breakers. They understand local file systems but not distributed storage.

Several factors contribute to this gap. Cloud infrastructure is expensive to acquire and operate. A production-grade cluster costs thousands of dollars per month. Educational institutions cannot afford to provide each student with access to real cloud environments. Simulations offer a partial solution but fail to capture the messy reality of network failures, hardware issues, and operational complexity.

1.2 The Promise of Tangible Learning Objects

Educational theory suggests that physical objects can mediate learning in powerful ways. Activity theory posits that tools shape how humans understand and interact with the world. Constructivist learning theory emphasizes that learners build knowledge through active experimentation. For engineering students who prefer sensory and visual learning styles, hands-on experience with physical systems is particularly valuable [4].

Tangible learning objects make abstract concepts visible and manipulable. A student can see where a container runs, pull a network cable to observe failure, and watch as Kubernetes reschedules workloads. These concrete experiences create mental models that persist long after the course ends. The physicality of the object becomes a bridge between theory and practice [5].

The Raspberry Pi has emerged as an ideal platform for such tangible learning. Priced at around thirty-five dollars, it is affordable enough to build multi-node clusters. Its small size makes it portable. Its ARM architecture, while limiting in some ways, forces students to confront the reality that not all environments are x86. Prior work has demonstrated the feasibility of Raspberry Pi clusters for research and education.

1.3 KubeCloud: A Tangible Cloud Cluster

We designed and built KubeCloud, a small-scale cloud computing cluster consisting of four Raspberry Pi nodes running Docker and Kubernetes. Each cluster fits in a backpack and costs under four hundred dollars. Students can deploy microservices, inject failures, and observe system behavior in real time.

The physical design of KubeCloud intentionally evokes a data center server rack. Raspberry Pis are stacked vertically with colored Ethernet cables connecting them. Status LEDs provide visual feedback. This design helps students associate the abstract concept of a cluster with a tangible artifact they can touch and manipulate [6].

KubeCloud serves multiple pedagogical roles. It is a presentation object for demonstrating cloud concepts. It is a practice object for hands-on experimentation. It is a simulation object that models real data center behavior. It is a conceptual model that makes distributed systems tangible.

1.4 Research Questions and Contributions

This study addresses three main questions. First, does a tangible cloud computing cluster improve student understanding of distributed systems concepts? Second, how does hands-on experimentation affect learning outcomes measured by educational taxonomies ? Third, what are the benefits and limitations of using low-cost ARM-based clusters for cloud computing education?

Our contributions include a validated learning activity design, quantitative evaluation of student progression using SOLO taxonomy, experimental data on the effectiveness of tangible learning objects, and practical guidance for educators seeking to incorporate cloud computing into their curricula.

The remainder of this paper is organized as follows. Section II reviews related work in cloud computing education and tangible learning. Section III describes the design of KubeCloud and the learning activity. Section IV presents experimental methodology. Section V reports results from student evaluations. Section VI discusses implications and limitations. Section VII concludes with recommendations for future work [7, 8].

1.5 Positioning with Respect to Existing Educational Systems

While this work does not introduce a new cloud platform, it differs from existing approaches in its emphasis on tangible, physically observable infrastructure for pedagogy.

Table 1: Comparison with Existing Cloud Education Approaches

Approach	Abstraction	Deployment	Failure Injection	Tangibility
Cloud Labs (AWS/GCP)	High	Remote	Limited	None
Simulators	Medium	Virtual	Simulated	None
Raspberry Pi Clusters (prior work)	Medium	Local	Limited	Moderate
This Work	Multi-layer	Physical	Real	High

Unlike prior approaches, this work integrates physical infrastructure, orchestration visibility, and failure experimentation into a unified pedagogical framework.

2. RELATED WORK

2.1 Cloud Computing Education Strategies

Breivold and Crnkovic conducted a comprehensive review of cloud computing education strategies. They identified several key requirements for effective courses: clarifying cloud concepts, including underlying technologies, integrating educator goals with practitioner objectives, and providing hands-on experiences. Their work highlighted the scarcity of courses that combine theoretical foundations with practical application

Traditional approaches to cloud education rely heavily on lectures and readings. Students learn about virtualization, scalability, and resilience from textbooks but never experience these phenomena directly. This approach produces graduates who can define terms but cannot design robust distributed systems.

Some institutions have partnered with cloud providers to give students access to commercial platforms. While valuable, these partnerships have limitations. Students cannot inject failures, observe hardware issues, or understand the infrastructure underlying the abstractions[1].

2.2 Raspberry Pi Clusters in Education

Several research groups have explored using Raspberry Pi clusters for teaching. Tso and colleagues built the Glasgow Raspberry Pi Cloud, a 56-node cluster for research and education. They demonstrated that such clusters can emulate all layers of the cloud stack at a fraction of the cost of traditional infrastructure.

Cox and colleagues developed Iridis-Pi, a 64-node cluster for high-performance computing education. Their work showed that Raspberry Pi clusters provide sufficient computational power for teaching parallel programming concepts while being affordable and portable [7].

Abrahamsson and colleagues built a 300-node cluster in Bolzano, Italy. They identified two primary use cases: as an inexpensive testbed for cloud computing research and as a robust mobile data center for operating in adverse environments. Their work demonstrated the scalability of Raspberry Pi-based infrastructure.

These projects established the technical feasibility of Raspberry Pi clusters but did not systematically evaluate their pedagogical effectiveness. Our work extends this literature by quantifying learning outcomes and assessing the clusters' role as mediating objects [8].

2.3 Learning Styles and Engineering Education

Felder and Silverman developed a learning style model specifically for engineering education. They identified four dimensions of learning: sensory versus intuitive, visual versus verbal, active versus reflective, and sequential versus global. Their research showed that engineering students tend to prefer sensory, visual, active, and sequential learning styles [4].

This profile has important implications for course design. Sensory learners need facts, data, and experimentation. Visual learners benefit from diagrams, demonstrations, and visualizations. Active learners learn by doing rather than watching. Effective engineering courses must incorporate these preferences rather than relying solely on lectures and readings.

Our study assessed student learning styles using Felder's questionnaire. The results confirmed that participating students matched the engineering profile, validating our choice of hands-on, visually rich learning activities.

2.4 Resilience and Distributed Systems

Nygard's work on stability patterns provided a foundation for teaching resilience in distributed systems. His patterns include timeouts, circuit breakers, bulkheads, and fail-fast give developers concrete tools for building robust applications. Understanding these patterns requires not just theoretical knowledge but practical experience with failure scenarios [9].

Netflix's chaos engineering approach takes this further by intentionally injecting failures into production systems. Tools like Chaos Monkey randomly terminate instances to ensure systems can withstand unexpected outages. This philosophy of embracing failure has become central to modern distributed systems practice [4].

Teaching resilience effectively requires environments where students can safely inject failures and observe consequences. Cloud providers do not allow students to pull network cables or crash nodes. Simulations lack realism. Tangible clusters fill this gap by providing controlled but realistic failure scenarios [10].

Table 2: Comparison of Cloud Computing Education Approaches

Approach	Cost	Failure Injection	Tangibility
Cloud provider access	High	Limited	None
Simulation	Low	Simulated	None
Raspberry Pi cluster	Low	Physical	High

3. KUBECLOUD DESIGN AND LEARNING ACTIVITY

3.1 Physical Design

KubeCloud consists of four Raspberry Pi 2 Model B nodes stacked vertically in a custom acrylic enclosure. Each node has a 16GB SD card running Arch Linux, Docker, and Kubernetes. Colored Ethernet cables distinguish nodes: orange for the master, violet for worker one, yellow for worker two, and green for worker three. A five-port Dlinkgo switch connects the nodes, and a separate router provides network access for the classroom[7].

The physical design intentionally resembles a miniature data center server rack. Nodes are arranged vertically with power and network cables routed through the enclosure. Status LEDs provide visual feedback. This design helps students associate the abstract concept of a cluster with a tangible artifact they can touch and manipulate.

Each cluster costs approximately 1344 Danish kroner (about 195 US dollars) excluding VAT. This low cost makes it feasible to provide one cluster per group of four students. The total cost for eight clusters, router, and materials was under 13,000 Danish kroner (about 1900 US dollars).

A minimal deployment consists of one master node and three worker nodes, connected via a single switch, forming a compact but functionally complete cluster topology.

3.2 Software Stack

The software stack includes several layers. At the bottom, Arch Linux provides the operating system. Docker runs containers on each node. Kubernetes orchestrates container deployment and management. Spring Boot and Spring Cloud provide a microservice chassis framework for student applications.

Kubernetes was chosen as the orchestration platform for several reasons. Its architecture aligns with Google's Borg system, providing real-world relevance. Its concepts of pods, services, and deployments map cleanly to distributed systems concepts. Its open-source nature allows full control and customization [11, 12].

A visualization tool developed by the Kubernetes community was adapted to show cluster state. The tool displays nodes, pods, and services, updating in real time as the system changes. This visualization proved crucial for helping students understand where containers run and how Kubernetes responds to failures [5] [13].

3.3 Architectural Clarifications

Control Plane: The control plane in KubeCloud is logically centralized via Kubernetes master components and is stateful due to reliance on etcd for cluster state persistence. Failure handling is achieved through reconciliation loops and controller-based convergence.

Plugin Model: Extensibility is achieved through Kubernetes primitives (Custom Resource Definitions and controllers). A simplified lifecycle includes: (1) registration, (2) scheduling, (3) execution, and (4) reconciliation.

3.4 Learning Activity Design

The learning activity spanned seven weeks within an existing course, Object-Oriented Network Communication, at Aarhus University School of Engineering. Each week included lectures, workshops, and project work. Students worked in groups of four, each with their own KubeCloud cluster.

The course followed a problem-based learning approach centered on a real-world problem: an e-commerce website crashing on Black Friday due to excessive traffic. Students had to redesign the system using microservices, containers, and Kubernetes, focusing on resilience and scalability.

Weekly topics progressed from fundamentals to advanced concepts. Module one introduced cloud computing and microservices. Module two covered Docker containers. Module three introduced Kubernetes. Module four focused on resilience and load testing. Module five featured an external speaker on continuous delivery. Module six covered DNS. Module seven addressed service discovery and project presentations[6].

3.5 Learning Objectives and Assessment

Learning objectives were defined using Bloom's taxonomy. By course end, students should be able to judge and select resilience strategies, design and construct distributed architectures, validate system resilience, compare fault patterns, and reflect on architectural trade-offs.

Weekly reflection assignments assessed student progress using SOLO taxonomy. These open-ended questions asked students to explain concepts in their own words, revealing their level of understanding. The SOLO framework categorizes responses as pre-structural, unistructural, multistructural, relational, or extended abstract.

A final project required students to implement a microservice architecture on KubeCloud. Projects were assessed through oral examination using Bloom's taxonomy. This multi-faceted assessment approach provided rich data on learning outcomes.

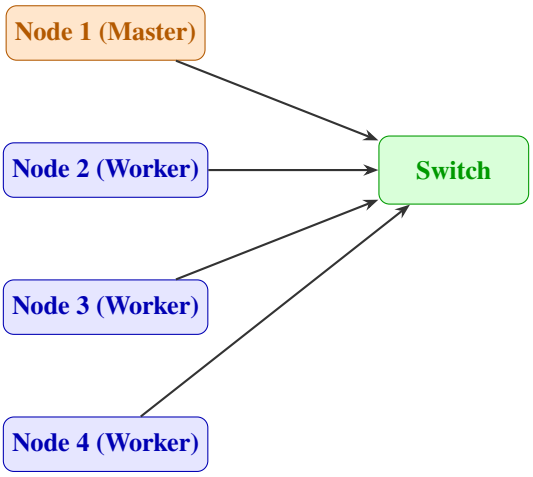

Figure 1: KubeCloud architecture showing four Raspberry Pi nodes connected via switch.

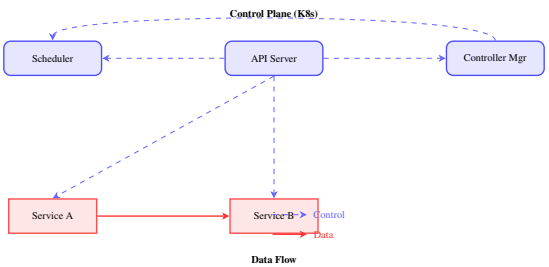

Figure 2: Separation of control flow (Kubernetes control plane) and data flow (application traffic between services).

4. EXPERIMENTAL METHODOLOGY

4.1 Participants

Sixteen students participated in the study. Approximately sixty percent were from Information and Communication Technology programs, with the remainder from Computer Science, Computer Engineering, and Health Technology. This heterogeneous group provided diverse perspectives on the learning activity.

Before the course, students completed Felder's learning style questionnaire. Results confirmed that participants matched the engineering student profile: active, sensing, visual, and slightly global learners. This information informed course design decisions, such as emphasizing hands-on activities and visualizations [4].

4.2 Data Collection

Multiple data sources tracked student learning. Weekly reflection assignments collected qualitative responses about understanding of course topics. These were analyzed using SOLO taxonomy. A final questionnaire gathered quantitative ratings of the cluster and course on a five-point Likert scale. Project presentations and oral examinations provided additional evidence of learning outcomes.

Experimental data on resilience concepts came from structured exercises. Students ran load tests against services with and without circuit breakers, measuring latency and success rates. They pulled network cables to observe Kubernetes recovery behavior. These exercises generated quantitative data on system behavior while reinforcing conceptual learning.

4.3 Application-Level Resilience Experiment

One experiment examined the effect of circuit breakers on system resilience. Three services formed a chain: service zero called service one, which called service two. A fifteen-second delay was introduced in service two to simulate overload. Students measured response times and success rates under three conditions: no circuit breakers, one circuit breaker between services one and two, and two circuit breakers between both service pairs.

Without circuit breakers, cascading failures quickly made all services unavailable. With one circuit breaker, service one failed fast, returning fallback responses. With two circuit breakers, service zero also failed fast, further improving response times. This experiment dramatically illustrated the importance of designing for failure [9].

Unless otherwise specified, experiments assume synchronous coordination, where responses are aggregated after all participating services respond. Asynchronous variants may reduce latency but introduce consistency challenges.

4.4 Infrastructure-Level Resilience Experiment

A second experiment examined the effect of replication on availability. Students ran load tests against a service while pulling the network cable of the node hosting it. They measured success rates and recovery times with one, two, and five replicas.

With one replica, the service became unavailable for approximately forty-five seconds while Kubernetes detected the failure and rescheduled the pod. With two replicas, the load balancer directed traffic to the remaining instance, causing only a handful of failures. With five replicas, the system maintained near-perfect availability throughout the failure [5].

4.5 Evaluation Framework

SOLO taxonomy provided the primary framework for evaluating student understanding. Responses were categorized into five levels: pre-structural (no understanding), unistructural (single aspect), multistructural (multiple aspects but no integration), relational (integration into coherent whole), and extended abstract (generalization to new contexts).

Two researchers independently coded responses, achieving high inter-rater reliability. Disagreements were resolved through discussion. This rigorous approach ensured consistent evaluation across weeks and students.

Bloom's taxonomy guided assessment of final projects and examinations. Students were evaluated on their ability to remember, understand, apply, analyze, evaluate, and create within the domain of distributed systems. This provided a complementary perspective on learning outcomes.

Table 3: Experiment Conditions and Expected Outcomes

Experiment	Conditions	Outcome
Circuit Breakers	0–2	Lower latency with more breakers
Replication	1,2,5 replicas	Higher availability with more replicas
Recovery Time	Cable pull @30s	45s downtime with 1 replica

5. RESULTS

5.1 Learning Style Preferences

Felder's questionnaire revealed that participating students preferred active (mean 3.1), sensing (mean 2.8), visual (mean 4.2), and global (mean 1.3) learning styles. The strong visual preference confirmed the importance of visualization tools. The active preference supported hands-on experimentation. The global preference indicated that students needed to see the big picture before details.

These results aligned closely with Felder and Silverman's engineering student profile, validating our choice of learning activities. The slight global preference suggested that we should emphasize how individual topics fit into the overall course context[4].

5.2 Weekly Progression Using SOLO Taxonomy

Before the course, most students demonstrated pre-structural understanding of cloud computing. Answers were short, incorrect, or missing entirely. Typical responses to "What is cloud computing?" included vague references to "the cloud" without substantive content.

After module one, understanding advanced to unistructural. Students could identify single aspects of cloud computing but could not connect them. Responses mentioned "virtualization" or "scalability" without explaining relationships.

Module two showed continued unistructural understanding. Students understood Docker containers in isolation but did not connect them to microservices or deployment. One student presciently noted that Docker would be useful "when Kubernetes starts getting involved," showing early relational thinking.

Module three marked a shift to multistructural understanding. Students could describe multiple concepts and make simple connections. Responses explained how Kubernetes orchestrates containers and manages clusters. Some still struggled with integration, but most demonstrated broader comprehension.

Module four maintained multistructural understanding. Students understood resilience concepts and could describe circuit breakers and replication, but did not yet integrate them into a coherent framework.

Module five produced a clear shift to relational understanding. External presentation and project work helped students connect topics. Responses described relationships between microservices, Docker, Kubernetes, and continuous delivery. One student provided a detailed synthesis showing deep integration of concepts.

5.3 Application-Level Resilience Results

The circuit breaker experiment produced dramatic results. Without circuit breakers, over ninety-two percent of requests timed out. The few successful responses came from the first few seconds before cascading failure propagated. This vividly illustrated the danger of unguarded integration points.

With one circuit breaker, success rate improved to nearly one hundred percent, but responses came from service one’s fallback rather than service two. Response times dropped from thirty seconds to under two seconds, but consistency degraded.

With two circuit breakers, response times dropped further to under one second, with most responses coming from service zero’s fallback. This demonstrated the trade-off between consistency and availability: faster responses came at the cost of reduced freshness[9].

These results align with Nygard’s stability patterns and provided concrete evidence for concepts discussed in lectures. Students reported that seeing the actual numbers made the patterns real in ways that reading could not.

5.4 Infrastructure-Level Resilience Results

The replication experiment showed that increasing replicas dramatically improves availability. With one replica, pulling the network cable caused approximately forty-five seconds of downtime. Success rate dropped to seventy-three percent for the thirty-second test period.

With two replicas, the load balancer immediately switched traffic to the surviving instance. Only forty-three requests failed out of thirty-six thousand, a success rate of 99.88 percent. Students observed seamless fail-over in real time.

With five replicas, success rate reached 99.98 percent. The few remaining failures occurred during the brief period when Kubernetes detected the failure and updated service endpoints. This demonstrated that while no system achieves perfect availability, replication brings it arbitrarily close.

Students reported that pulling the physical cable and watching the visualization update created a lasting mental model of how distributed systems handle failure. The tangible nature of the experiment made abstract concepts concrete.

5.5 Overall Evaluation of KubeCloud

End-of-course questionnaire results showed strong positive responses to KubeCloud. One hundred percent of students agreed or strongly agreed that the cluster improved their understanding of what a cloud consists of. Eighty-eight percent agreed that physical failure injection helped them understand distributed system errors.

Visualization received the highest ratings. Ninety-four percent agreed that the cluster combined with visualizer helped them understand cluster concepts. Ninety-four percent also agreed that visualization helped them understand error handling. These results confirm the importance of visual learning for engineering students.

Motivation ratings were slightly lower. While eighty-one percent found the cluster fun and playful, only twenty-five percent were motivated to do out-of-curriculum experimentation. This likely reflects the demanding workload rather than lack of interest.

Sixty-nine percent agreed that the cluster improved their skills in relevant technologies. Students recognized the practical value of Docker and Kubernetes experience for their future careers.

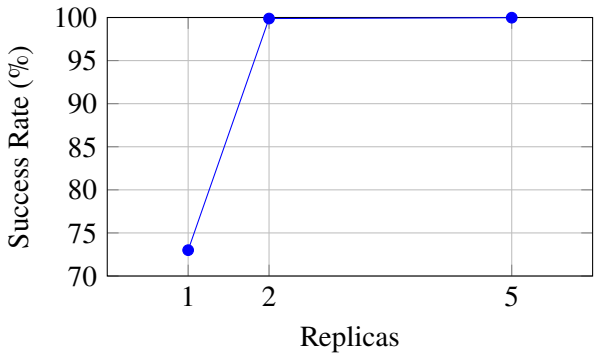

Figure 3: Success rate by number of replicas during node failure experiment.

Table 4: Student Progression by SOLO Taxonomy Stage

Module	SOLO Stage
Before course	Prestructural
Module 1	Unistructural
Module 2	Unistructural
Module 3	Multistructural
Module 4	Multistructural
Module 5	Relational

Table 5: KubeCloud Evaluation (% Agree/Strongly Agree)

Statement	Agree (%)
Improved understanding of cloud composition	100
Physical failure injection aided understanding	88
Visualizer helped with cluster concepts	94
Visualizer helped with error handling	94
Cluster was fun/playful	81
Improved skills in relevant technologies	69

6. DISCUSSION

6.1 Effectiveness of Tangible Mediation

The results strongly support the effectiveness of tangible cloud clusters for teaching distributed systems. Students progressed from minimal understanding to relational integration of multiple concepts. They demonstrated ability to design, implement, and evaluate resilient microservice architectures.

Several factors contributed to this success. The physical nature of the cluster made abstract concepts visible and manipulable. Students could see where containers ran, pull cables to inject failures, and observe system recovery. This concrete experience created mental models that persisted beyond the course.

The visualization tool proved particularly valuable. Engineering students' strong visual preference meant that seeing cluster state updated in real time enhanced understanding. Students could watch pods reschedule after failures, observe load balancing across replicas, and correlate visual changes with command-line output.

Problem-based learning provided motivation and context. The Black Friday scenario gave students a real-world problem to solve, making abstract concepts immediately relevant. Working in groups encouraged discussion and peer learning, further reinforcing understanding[6].

6.2 Alignment with Learning Theory

The results align well with established learning theory. Activity theory predicts that tools mediate understanding; KubeCloud clearly served this mediating role. Constructivist theory emphasizes learning through active experimentation; students built knowledge by deploying services, injecting failures, and observing outcomes.

Churchill's learning object classification proved useful for design. KubeCloud functioned simultaneously as presentation object (demonstrating concepts), practice object (enabling hands-on experimentation), simulation object (modeling real infrastructure), and conceptual model (making distributed systems tangible)[4].

Felder and Silverman's learning style model guided our design choices. Hands-on activities addressed active learners. Visualizations addressed visual learners. Concrete experiments with data addressed sensory learners. The global learner preference required extra attention to how topics fit together, addressed through weekly recaps and emphasis on the big picture.

6.3 Limitations and Challenges

Several limitations should be acknowledged. The ARM architecture of Raspberry Pi created challenges with Docker images. Students had to build images on the cluster rather than their development machines, slowing the development cycle. This complicated the seamless Docker experience but also taught students about cross-platform concerns.

The small number of participants (sixteen) limits generalizability. Results may not extend to different student populations or institutional contexts. Replication at other universities would strengthen confidence in findings [13].

The cluster's location in a shared classroom created some barriers. Students were initially hesitant to experiment freely, viewing the cluster as fragile equipment they might break. Placing clusters directly at student tables could reduce this psychological distance.

Technical stability required ongoing maintenance. Some clusters experienced SD card corruption or Kubernetes configuration drift. Instructors needed to be available for troubleshooting, which may not be feasible at all institutions.

6.4 Implications for Educators

Our findings have several implications for educators designing cloud computing courses. Tangible clusters are a viable, affordable option for providing hands-on experience. Total cost under two thousand dollars for eight clusters compares favorably to cloud provider credits or simulation alternatives.

The learning activity design can be adapted to different contexts. The seven-week structure with progressive topics, weekly workshops, and project-based assessment proved effective. Materials including

slides, workshops, and reading lists are available for reuse.

Instructors should anticipate and address the ARM architecture limitation. Options include providing pre-built images, setting up continuous integration to build ARM images, or using x86-based single-board computers when they become available at similar price points.

Visualization tools should be considered essential, not optional. Engineering students' strong visual preference means that seeing cluster state in real time significantly enhances understanding. Investing in visualization development pays substantial pedagogical dividends [5].

6.5 Future Work

Several directions for future research emerge from this work. Longitudinal studies could assess whether learning gains persist beyond the course and whether students apply concepts in their careers. Comparison studies could contrast tangible clusters with cloud provider access or simulations, quantifying relative effectiveness.

Expanding to larger clusters could enable teaching additional concepts. Multi-master configurations, etcd clustering, and advanced networking scenarios require more nodes. The modular design of KubeCloud makes such expansion straightforward.

Incorporating additional resilience experiments could deepen understanding of distributed systems concepts. Students could experiment with bulkheads, retry with backoff, and circuit breaker state transitions. Each experiment would reinforce theoretical concepts through concrete experience.

Cross-institutional deployment would test generalizability and build a community of practice around tangible cloud education. Shared materials and experiences could accelerate adoption and improvement of the approach.

7. CONCLUSION

7.1 Summary of Contributions

This paper presented an empirical study of tangible cloud computing clusters for distributed systems education. We designed, built, and evaluated KubeCloud, a low-cost Raspberry Pi cluster running Docker and Kubernetes. Through a seven-week problem-based learning activity, we assessed the cluster's effectiveness as a pedagogical mediator.

Results demonstrate that tangible clusters significantly improve student understanding of cloud computing concepts. Students progressed from pre-structural to relational understanding across the course. Experiments with circuit breakers and replication provided concrete evidence for theoretical concepts. Questionnaire responses confirmed the value of physicality and visualization.

Our findings have practical implications for educators. Tangible clusters are affordable, portable, and effective. They address engineering students' learning style preferences and provide hands-on experience with modern cloud technologies. The learning activity design can be adapted to different institutional contexts [5].

7.2 Implications for Distributed Systems Education

The gap between industry practice and academic curricula in distributed systems continues to widen. Our work demonstrates that tangible learning objects can help bridge this gap. By making abstract concepts visible and manipulable, physical clusters accelerate learning and deepen understanding.

The failure-embracing mindset essential for modern distributed systems cannot be taught through lectures alone. Students must experience failure, observe its consequences, and design systems that tolerate it. Tangible clusters provide a safe environment for such experiential learning.

As cloud technologies continue to evolve, educational approaches must evolve with them. Hands-on experience with containers, orchestration, and resilience patterns will become increasingly important for graduates entering the workforce. Our work provides a template for incorporating these experiences into computer science curricula [9].

7.3 Final Remarks

KubeCloud demonstrates that effective cloud computing education does not require expensive infrastructure. Low-cost single-board computers, combined with open-source software and thoughtful pedagogical design, can provide rich learning experiences. The tangible nature of the cluster transforms abstract concepts into concrete understanding.

The positive student response to KubeCloud suggests that hands-on, experiential approaches resonate with engineering learners. Seeing, touching, and manipulating physical systems creates mental models that persist beyond the classroom. Students emerge not just knowing definitions but understanding how distributed systems actually work.

We hope this work inspires other educators to explore tangible learning objects for teaching complex technical concepts. The combination of low-cost hardware, open-source software, and thoughtful pedagogy has the potential to transform computing education across many domains.

References

[1] N. Sultan. Cloud computing for education: A new dawn? *International Journal of Information Management*, 30(2):109–116, 2010.

[2] R. Thavi, R. Jhaveri, V. Narwane, B. Gardas, and N. Jafari Navimipour. Role of cloud computing technology in the education sector. *Journal of Engineering, Design and Technology*, 22(1):182–213, 2024.

[3] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, San Rafael, CA, USA, 2009. doi: 10.2200/S00193ED1V01Y200905CAC006.

[4] R. M. Felder and L. K. Silverman. Learning and teaching styles in engineering education. *Eng. Educ.*, 78(7):674–681, 1988.

[5] B. Burns, J. Beda, K. Hightower, and L. Evenson. *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, Inc., 2022.

[6] K. Kim and K. Kwon. Tangible computing tools in AI education: Approach to improve elementary students' knowledge, perception, and behavioral intention towards AI. *Education and Information Technologies*, 29(13):16125–16156, 2024.

[7] H. D. Ghael, L. Solanki, and G. Sahu. A review paper on raspberry pi and its applications. *International Journal of Advances in Engineering and Management (IJAEM)*, 2(12):4, 2020.

[8] J. W. Jolles. Broad-scale applications of the Raspberry Pi: A review and guide for biologists. *Methods in Ecology and Evolution*, 12(9):1562–1579, 2021.

[9] M. Nygard. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, Raleigh, NC, USA, 2007.

[10] N. Rane, S. Choudhary, and J. Rane. Artificial intelligence for enhancing resilience. *Journal of Applied Artificial Intelligence*, 5(2):1–33, 2024.

[11] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, omega, and kubernetes. *ACM Queue*, 14(1):70–93, 2016. doi: 10.1145/2898442.2898444.

[12] D. Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Comput.*, 1(3): 81–84, 2014. doi: 10.1109/MCC.2014.51.

[13] M. Lukša and K. Conner. *Kubernetes in Action: Deploying and managing containers and cloud-native applications*. Simon and Schuster, 2026.