IntentContinuum: Using LLMs to Support Intent-Based Computing Across the Compute Continuum

Negin Akbari*, John Grundy*, Aamir Cheema*, Adel N. Toosi*†
*Department of Software Systems and Cybersecurity, Monash University, Australia,
{negin.akbari, john.grundy, aamir.cheema, adel.n.toosi}@monash.edu
†School of Computing and Information Systems, The University of Melbourne, Australia,
adel.toosi@unimelb.edu.au

***Abstract*—The increasing proliferation of IoT devices and AI applications has created a demand for scalable and efficient computing solutions, particularly for applications requiring real-time processing. The *compute continuum* integrates edge and cloud resources to meet this need, balancing the low-latency demands of the edge with the high computational power of the cloud. However, managing resources in such a distributed environment presents challenges due to the diversity and complexity of these systems. Traditional resource management methods, often relying on heuristic algorithms, struggle to manage the increasing complexity, scale, and dynamics of these systems, as well as adapt to dynamic workloads and changing network conditions. Moreover, designing such approaches is often time-intensive and highly tailored to specific applications, demanding deep expertise. In this paper, we introduce a novel framework for intent-driven resource management in the compute continuum, using large language models (LLMs) to help automate decision-making processes. Our framework ensures that user-defined intents – such as achieving the required response times for time-critical applications – are consistently fulfilled. In the event of an intent violation, our system performs root cause analysis by examining system data to identify and address issues. This approach reduces the need for human intervention and enhances system reliability, offering a more dynamic and efficient solution for resource management in distributed environments.**

Keywords- Compute Continuum, Resource Management, Intent-driven scheduling, LLM.

I. Introduction

The rapid growth of the AI-driven Internet of Things (IoT) has led to a massive increase in smart devices, each generating large amounts of data [1]. This surge requires scalable storage and processing solutions, such as cloud computing. However, cloud computing alone may not be suitable for applications that need real-time processing or strong privacy protections [2]. This challenge has led to the development of the *"compute continuum"* that combines edge and cloud resources to ensure smooth and efficient operations across a wide range of applications [3]. In this integrated system, edge devices often have limited computing power and storage, yet they must handle tasks that require low latency and quick response times [4]. On the other hand, cloud resources offer greater computational power and storage but involve higher latency and potential privacy concerns [5]. Resource management is thus particularly challenging due to the need to balance the diverse requirements of both edge and cloud environments [6]. Efficient resource management must account for these differences, ensuring that tasks are allocated optimally between the edge and cloud. This involves dynamically adjusting to changes in workload, network conditions, and energy consumption, all while maintaining the seamless operation of applications.

The compute continuum, with its diverse and distributed nature, poses significant challenges for resource management and scheduling, which are traditionally categorized as NP-hard in its general case [7]. Solving these problems optimally becomes computationally infeasible for large instances due to the exponential growth of possible solutions. To address this, heuristic [8] and meta-heuristic approaches [9], such as genetic algorithms (GA) [10] and simulated annealing [11], are widely used to find near-optimal solutions within a reasonable timeframe. However, designing efficient heuristic algorithms often requires domain expertise and problem-specific tuning, which limits their adaptability to the heterogeneous and dynamic demands of the compute continuum [6]. As a result, current efforts focus on automating these processes to reduce dependency on human intervention and specialized algorithms tailored to specific problems.

With the rapid advancements in artificial intelligence (AI) and the increasing prevalence of machine learning (ML) techniques, generative AI and large language models (LLMs) have achieved a level of intelligence comparable to that of human experts. In this paper, we explore whether "*LLMs can complement—or even replace—both human experts and the need for developing specialized algorithms, offering a more generalized and efficient approach for resource management in compute continuum*". We aim to leverage the capabilities of general purpose LLMs, such as ChatGPT, to develop novel intent-driven resource management techniques for distributed systems. We believe that LLMs, with their ability to process and analyze vast amounts of data, offer a promising solution to these challenges while simplifying the overall process. By leveraging the adaptive and contextual understanding capabilities of LLMs, resource management can be more dynamic

and responsive to the continuum's needs.

Thus, we present a novel framework, *IntentContinuum*, designed to manage and optimize application deployments and operations across the continuum, ensuring that user-defined intents—particularly service-level objectives (SLOs) for response times in image-processing IoT applications—are consistently met. Our framework uniquely monitors, analyzes, and addresses any deviations from these intents, thereby maintaining optimal performance across the compute continuum.

IntentContinuum uses the integration of LLMs, specifically Open AI GPT-4o, as a central decision-making entity within the framework. When a violation of the predefined intent occurs, our algorithm employs GPT-4o to conduct a comprehensive root cause analysis. By processing system data—including network topology, cluster information, and real-time monitoring metrics—GPT-4o determines whether the issue originates from computational constraints (such as CPU or memory shortages) or network-related problems (like bandwidth limitations or link congestion). Following the identification of the root cause, GPT-4o suggests specific actions to resolve the issue, drawing from a predefined list of potential solutions. These actions are automatically configured in the system, initiating a continuous feedback loop to ensure that the user-defined intent is always satisfied. This loop allows the system to adapt to changes in workload, network conditions, or other environmental factors, reducing the need for manual intervention and enhancing the reliability of the compute continuum environment. The **key contributions** of this work include:

- We propose *IntentContinuum*, an innovative monitoring and automated reconfiguration framework for the compute continuum, designed to support a range of user-specified intent-driven performance criteria;
- We propose an LLM-powered root cause analysis and automated reconfiguration platform for *IntentContinuum* to manage user intents in the compute continuum. To the best of our knowledge, we are among the first to leverage LLMs as resource managers in this way, with initial evaluations showing promising results;
- We develop a prototype of *IntentContinuum* using industry-standard techniques and release it as an open-source solution to demonstrate its effectiveness in real-world scenarios; and
- We conduct an extensive evaluation of the *IntentContinuum* platform's performance across various practical scenarios and user-defined intents, complemented by a preliminary feasibility study assessing its potential impact.

II. Motivation

Consider a real-time image-processing application for predictive maintenance in an Industry 4.0 setting, where strategically placed camera sensors capture data from production lines and equipment. These sensors work in tandem with edge and cloud components to analyze images and detect potential issues. Such an application must meet strict SLOs, like low response times for detecting faults, despite dynamic conditions such as fluctuating network latency or intermittent connectivity.

Ensuring these SLOs are consistently satisfied is a challenging and non-trivial task. The factory environment relies on a wide variety of computational resources across the edge-cloud continuum where the application is deployed, coupled with evolving network conditions and shifting operational priorities—for example, prioritizing real-time issue detection during peak hours versus energy efficiency during off-hours. Developing robust resource management techniques to dynamically adapt and ensure these SLOs are met is essential, as traditional rule-based approaches often fall short in handling such variability and unpredictability.

Thus, we aim to address the following key research questions:

RQ1: –"*Can a large language model (LLM), such as GPT-4o, enhance real-time resource management in the compute continuum for IoT applications?*"

RQ2: –"*How effective is such an LLM-powered approach in identifying and resolving issues when performance goals (e.g., response times) are violated in the compute continuum?*"

III. Our Approach

A. IntentContinuum Architecture

Figure 1 illustrates the architecture of our *IntentContinuum* framework, designed to manage and optimize application deployment and operations within a compute continuum environment. Below we describe its key components and their roles within the system. The complete source code is publicly available on our GitHub repository.[1]

Target Compute Continuum Environment: At the bottom of Figure 1, we illustrate the target compute continuum environment, where nodes, including both edge and cloud servers, are managed by a container orchestrator such as Kubernetes. Various microservices (pods in Kubernetes) run on these nodes, interconnected through network switches controlled by a Software Defined Network (SDN) controller. Each element in this continuum, including edge and cloud servers, network switches, and application pods—utilizing sidecar containers[2]—is continuously monitored by our monitoring tool. Requests from sensors or end users are directed to the application's ingress gateway for processing. Users can define specific SLOs for their application, such as response time targets or energy efficiency, based on high-level intents. In this context, intents represent high-level user goals, such as minimizing latency or optimizing energy consumption, while SLOs define measurable performance targets, such as maintaining an average response time below a specified threshold. For the purposes of this paper, intents and SLOs are used interchangeably.

The dynamic nature of this environment, as highlighted in our contributions, demands ongoing adaptation to meet these user-defined intents.

[1]https://github.com/disnetlab/IntentContinuum

[2]https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/

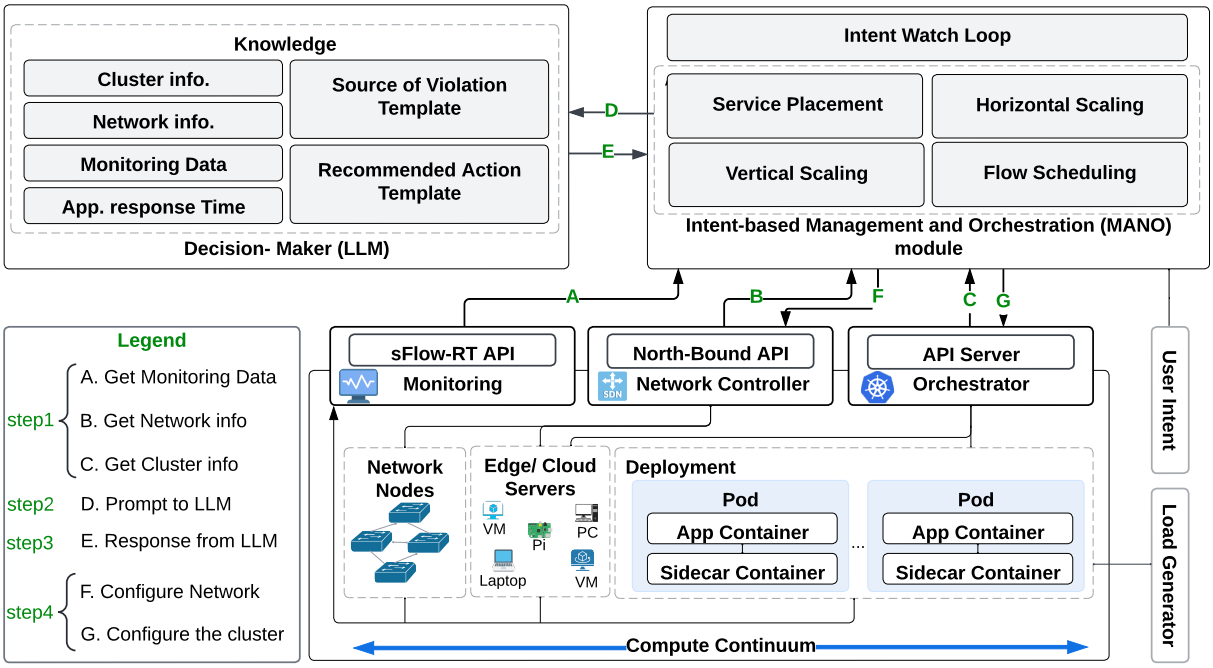


Fig. 1: Architecture of our *IntentContinuum* system

Intent-based Management and Orchestrator (MANO) Module: An *Intent Watch Loop* continuously monitors the user-defined intent, which is provided in JSON format. An example is shown below, where the intent specifies lower and upper thresholds for the application's response time, set to 2 and 4 seconds, respectively. If the actual response time exceeds the upper threshold or falls below the lower threshold, it may indicate a violation of the defined intent. The system performs this monitoring across the compute continuum, detecting any such potential violations for the target application..

```
{
  "intent": {
    "Min response_time(s)": 2,
    "Max response_time(s)": 4
  }
}
```

Upon detecting a violation, the watch loop triggers the *Decision-Maker* module to resolve the issue. The *Monitoring* module, in the target compute continuum, retrieves information on application performance, network status, and cluster details through concurrent API calls (step 1 in the figure). This data helps create a clear picture of the system's current state and identify deviations from the user-defined intents.

The collected data is then sent to the *Decision-Maker* module via an API call, where the root cause of the violation is diagnosed (step 2). Based on the analysis, appropriate corrective actions are recommended to resolve the issue and ensure compliance with the intents (step 3). Validated by recent research in cloud-edge computing [12], [13], [14], [15], we consider four corrective actions: 1) *service placement* (relocating pods to a more suitable node to optimize resource utilization and minimize latency), 2) *horizontal scaling* (adjusting the number of pod replicas to handle dynamic workloads efficiently), 3) *vertical scaling* (modifying CPU and memory allocations to enhance stability and resource efficiency), and 4) *flow scheduling* (rerouting application network traffic through optimal paths to mitigate congestion and network instability). By integrating these approaches, our method ensures improved application responsiveness and resource allocation in the compute continuum. Finally, the Network Controller and Orchestrator apply the necessary reconfigurations on the relevant nodes via their APIs (step 4) to address the root causes of intent violation.

Decision-Maker: The core of the system is the *Decision-Maker* module, powered by a large language model (LLM), specifically OpenAI GPT-4o in this work. This module processes data collected from various system components and orchestrates responses to maintain system performance. The *Decision-Maker* begins by receiving a comprehensive narrative that describes the system's architecture and operational dynamics, providing essential context for GPT-4o. The collected data is formatted as JSON objects (Figure 2(a)), which encapsulate three primary categories: (1) *Cluster information*, detailing nodes and pods along with their CPU and memory allocations; (2) *Network information*, representing hosts, switches, ports, and interconnecting links; and (3) *Monitoring data*, which tracks real-time resource utilization at the node, pod, and network link levels. These structured inputs ensure a comprehensive and machine-readable representation of the system's state, enabling accurate root cause analysis. When a deviation from expected performance is detected (indicating an intent violation), the *Decision-Maker* identifies the root cause by analyzing the collected data using a structured *Source of Violation Template* (Figure 2(b)). These sources of violations are the major performance challenges reported in cloud and

edge computing [16], [14]. Following this analysis, the module leverages a *Recommended Action Template* (Figure 2(c)) to propose targeted corrective actions, addressing issues related to computational resources, network conditions, or other operational factors. To enhance GPT-4o's decision-making, the prompt is structured using a template-based approach and few-shot learning [17]. Few-shot learning is incorporated directly within the prompt, providing structured examples of intent violations, root cause analyses, and corrective actions. This approach ensures consistent and optimized decision-making without modifying the underlying model. The *Decision-Maker* integrates its recommendations with the *MANO* module, enabling seamless implementation of corrective measures.

B. Detailed Decision Making Process

Figure 3 illustrates the sequence diagram of our *IntentContinuum* tool decision making process, highlighting the interactions between each module when a defined intent is violated. The process begins with the *Intent Watch Loop*, a continuous monitoring system that detects intent violations, such as when the application response times exceed a specific threshold (e.g., smaller than 6 seconds). Upon detecting a violation, the system initiates corrective actions.

In the first stage, *MANO* makes an API call to the *Orchestrator* to gather cluster information, including the state and configuration details of the applications and services running within the cluster. Concurrently, an API call is made to the *SDN Controller* to obtain network information, detailing the status and configuration of network resources. Another concurrent API call is directed to the *Monitoring* module to collect comprehensive monitoring data, such as CPU and memory utilization of nodes and pods in the cluster, as well as the amount of traffic on each switch interface. These API calls are collectively labeled as 1 in the sequence diagram for reference. Given the large volume of monitoring data, we condense the information before sending it to the *LLM* by aggregating average metrics over multiple segments: 'pre-violation' and 'violation.' These segments are discussed in more detail later.

Response Time SLO: To maintain the response time SLO within the specified range and prevent the system from over-reacting to temporary fluctuations or noise, we calculate the *Exponential Moving Average (EMA)* of response time, which is represented as:

$$\text{EMA\_RT}_t = (1 - \alpha) \times \text{EMA\_RT}_{t-1} + \alpha \times \text{RT}_t \quad (1)$$

Here, RT_t denotes the response time of the most recent request at time t, and EMA_RT_t represents the exponential moving average of the response time at time t. The smoothing factor α which is in the range of (0,1] is defined by the system administrator based on the desired sensitivity to performance changes. In our implementation, we use α= 0.02, which provides a balance between responsiveness and stability—allowing the system to react to sustained trends while filtering out short-lived fluctuations or noise. This method continuously updates EMA_RT, assigning more weight to recent samples. A violation is detected when the EMA_RT exceeds or drops below predefined thresholds, marking a significant performance deviation. As soon as a violation is detected, the collected details are sent to *LLM* for analysis and find the appropriate actions to rectify the issue. This process is labeled as 2 in the sequence diagram for clarity.

For the 'pre-violation' segment, we analyze the system's response times across windows, e.g., last 30 requests. The pre-violation data is collected using a **Fixed-Time-Window Aggregation** method, meaning data is aggregated into fixed-length windows immediately preceding the violation. Mathematically, this is represented as:

$$\text{Pre-Violation} = \{..., \omega_{t-2}, \omega_{t-1}\}$$

where ω_{t-1}, ω_{t-2}, etc., represent the windows immediately preceding the violation, while the window ω_t contains the violation and represents the anomaly. This approach helps identify relevant system performance metrics at the time of the response time breach, providing insights into potential causes. By structuring the data this way, we provide *LLM* with a concise yet informative view of the system's performance both before and during the violation for further analysis.

Next, *IntentContinuum* sends prompts to the *LLM*, step 2 in the sequence diagram, which processes the collected data. The *LLM* analyzes the situation, identifies the source of the violation, and recommends corrective actions (step 3). Following the *LLM*'s recommendations, the *Orchestrator* updates the deployment, step 4 and step 5, by making the necessary adjustments to the applications or configurations. The *SDN Controller* also reconfigures the network to align with the updated deployment, step 4 and step 5. This way, our framework enables a dynamic, automated reactions to intent violations, ensuring system resilience and optimal performance.

IV. Performance Evaluation

In this section, we present a performance evaluation of *IntentContinuum* using an example image processing application as a representative scenario. We describe our experimental testbed setup in detail, followed by an in-depth analysis of the evaluation results. These results provide valuable insights into the performance and capabilities of our proposed *IntentContinuum* method using LLMs. We benchmark our approach against existing methods to highlight its effectiveness, advantages and limitations. Our results provide valuable insights into the performance and capabilities of our proposed *IntentContinuum* method using LLMs.

A. Experimental Setup

To evaluate *IntentContinuum*, we leverage *iContinuum* emulator [15], creating a controlled environment for studying the proposed intent-driven resource management. *iContinuum* offers a flexible platform for deploying real-world applications over the edge-cloud continuum and simulates practical conditions by integrating SDN controllers, Mininet-based network

```
{
  "Cluster_info": {
      "nodes": [{"name": "", "cpu": "", "mem": ""}],
      "pods": [{"name": "", "location": "", "cpu": "", "mem": ""}]
  },
  "Network_info": {
      "hosts": [{"ip": [""], " Switch ": "", "port": ""}],
      "links": [{"src": {"port": "", " Switch ": ""},
       "dst": {"port": "", " Switch ": ""}}]
      },
  "Monitoring_data": {
      "node": [{"cpu": ""}, {"mem": ""}],
      "pod": [{"cpu ": ""}, {"mem": ""}],
      "links": [{"out_utilization": ""}, {"in_utilization": ""}] } }
```

```
{
  "Source of Violation": {
      "Reason": [
          {"description": "high_cpu_utilization_on_nodes",
          "details": {"nodes": [""]}},
          {"description": "high_cpu_utilization_on_pods",
          "details": {"pods": [""]}},
          {"description": "high_mem_utilization_on_nodes",
          "details": {"nodes": [""]}},
          {"description": "high_mem_utilization_on_pods",
          "details": {"pods": [""]}},
          {"description": "traffic_congestion_on_switches",
          "details": {"switch": [""]}},
          {
          "description": "Unknown_Reason",
          "details": {
          "description": ["Description of the unknown reason
          or issue"] } } ] } }
```

```
{
  "Recommended Action": {
    "pods": [{
        "names": "",
        "new_cpu_limit": "",
        "now_mem_limit": "",
        "new_replicas": ""}],
    "new_placement": [
       {"pod": "", "node": ""} ],
    "new_traffic_path": {
        "send_path": [ {
            "src_node": "", "switch": [""] },
          {
            "dst_node": "" } ],
        "reverse_path": [
          {
            "src_node": "", "switch": [""] },
          {
            "dst_node": "" } ] } } }
```


(a) API Calls (b) Source of Violation Template (c) Recommended action Template

Fig. 2: Examples of JSON objects sent to the LLM in order to provide structured inputs and outputs

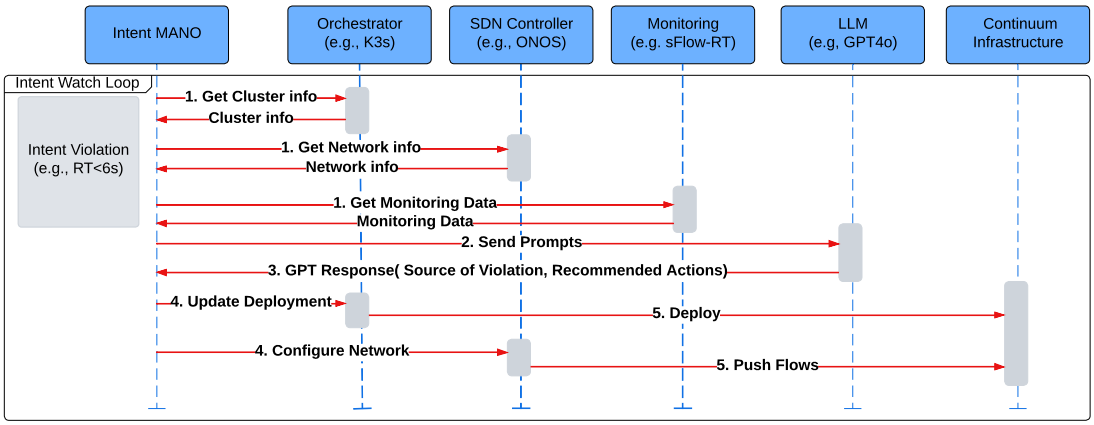


Fig. 3: Sequence Diagram of key *IntentContinuum* process steps

emulation, and containerized applications managed by Kubernetes. Figure 4 illustrates the topology of the experimental setup. We used a Kubernetes cluster consists of one Master node (M) and three Worker nodes (W1–W3), interconnected via Open vSwitches (OVS), *S1–S6* in the figure, which are managed by ONOS as the SDN controller. The application deployed on the Kubernetes cluster is an image processing application composed of a chain of four microservices, hosted on four pods (p1, p2, p3, and p4). These pods are distributed across the cluster nodes and communicate through the red dashed arrows in the figure, which represent the flow of data. Additionally, the Master node hosts a Load Generator (LG) that sends traffic to the pods and a Database (DB) that records pod activity. The green dashed arrow shows HTTP requests sent from Locust, acting as the Load Generator. Locust sends a 499.69 KB image to the entry microservice in the chain, hosted on pod p1. Detailed configuration of the nodes are presented in Table I.

Node	OS	Arch.	RAM	vCPU
Edge Servers	Ubuntu 20.04 LTS	amd64	64GB	32
SDN Controller	Ubuntu 20.04 LTS	amd64	64GB	32

TABLE I: Configuration of Nodes

We define the intent as a response time that must remain within a specified range, bounded by upper and lower thresholds, to ensure efficient resource utilization and consistent application performance. An intent violation is detected based on EMA_RT (as defined in Eq. (1)) when it either exceeds the upper threshold or drops below the lower threshold. For our experiments, we set the upper threshold to 3 seconds and the lower threshold to 1 second, determined through initial testing under varying traffic conditions. These thresholds represent a reasonable range for our application's operational tolerance while ensuring stability and responsiveness. This dual-threshold approach ensures a balance between cost-

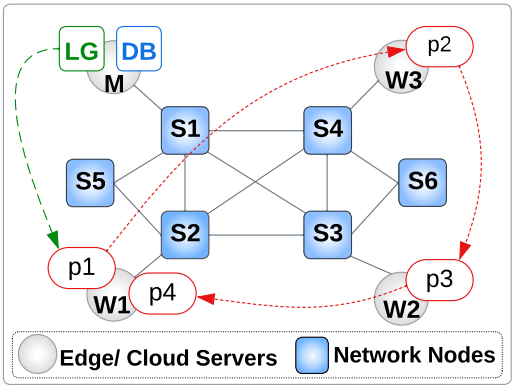


Fig. 4: Scenario's Network Topology

efficiency and performance. The upper threshold safeguards against performance degradation by flagging insufficient resource allocation, while the lower threshold prevents resource wastage by identifying over-provisioning. If the EMA_RT exceeds the upper threshold, the system flags a violation and sends details to GPT, which identifies the root cause and recommends corrective actions. Similarly, if the EMA_RT drops below the lower threshold, the system consults GPT to analyze the situation and recommend optimization strategies to ensure efficient resource usage.

To reflect the varying computational demands of each pod, resource limits are configured for the pods as presented in Table II. Since pod p3 has higher processing requirements compared to the other pods, it is assigned increased resource limits. Pod placement is initially managed by Kubernetes' default scheduler; however, pod p1 is fixed on Worker1 to serve as the first entry point of the application, ensuring consistent traffic flow into the system. The placement of the pods within the cluster is illustrated in Figure 4. The pods are managed behind Kubernetes services, which provides load balancing and facilitates communication between the pods and external systems. Additionally, the system is designed to support multiple replicas of the same pod to handle varying workloads and ensure high availability.

Pod name	CPU limit (core)	Mem limit (MiB)
p1, p2, p4	0.3	312
p3	0.5	512

TABLE II: Configuration of Pods

Traffic generation is managed using Locust for a duration of 900 seconds, simulating user loads with the following sequence: [10, 20, 15, 10, 5, 20, 10] users, with a spawn rate of 1 user per second. The interval between each load change is set to 120 seconds. During the experiment, if the system detects that the EMA_RT exceeds the maximum threshold or falls below the minimum threshold, it flags the violation and sends an API call to GPT to address the violation. After implementing these actions, the system introduces a "waiting-time" metric, defined as the duration during which the system halts monitoring and evaluation to allow the network to stabilize. For our setup, the "waiting time" is set to 1 minute after each corrective action, determined through experimental testing to ensure the system reaches a stable state before further evaluation and reconfiguration.

B. Results and Analysis

In this section, we present experiments demonstrating how *IntentContinuum* effectively satisfies the intent by addressing both computing and networking actions.

1) ***Computing Experiment****:* Figure 5 presents the application response time under varying load levels generated by Locust. The x-axis represents the time (in seconds) during the experiment, while the y-axis indicates the response time (in seconds) for both RT, indicating the response time for each individual request (dashed blue line), and EMA_RT (red line), calculated using the formula described in Eq. (1). The graph

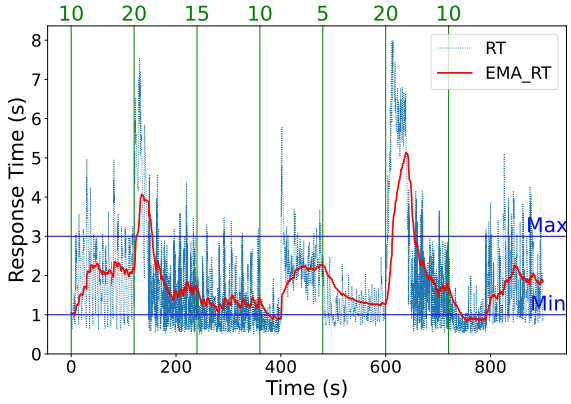


Fig. 5: Application Response Time for Computing Experiment

includes the upper (max) and lower (min) thresholds, set at 3 seconds and 1 second, respectively, offering a clear depiction of the application's performance relative to the defined intent boundaries. The green numbers displayed at the top of the graph represent the number of active users sending requests (system load).

At the start of the experiment, with 10 users, EMA_RT stays within the defined thresholds, as shown in the graph. After 120 seconds, the user count increases to 20, causing a sharp spike in EMA_RT that exceeds the upper threshold. At this point, *IntentContinuum* promptly identifies the source of the violation, which is high CPU consumption on the application pods, by analyzing data from both the pre-violation and violation windows, along with the corresponding monitoring metrics. This comprehensive view enables the system to accurately detect the root cause, and recommends corrective actions, successfully restoring the response time to within acceptable limits after about 30 seconds. A one-minute waiting period follows the corrective measures to ensure system stability. At 240 seconds, the user count decreases to 15, leading to a drop in EMA_RT. No violations occur during this period, as the previous corrective actions proved highly effective in maintaining system stability. Subsequently, at 360 seconds, the user count decreases further to 10, resulting in lower threshold intent violations—as the system maintains more pod resources than necessary for the current traffic load—around the time of 380 seconds. *IntentContinuum* quickly addresses this violation by scaling down/in unnecessary resources, ensuring the response time returns to within acceptable boundaries. When the user count drops to 5 at 480 seconds, the system maintains the desired response time without any violations. The load then increases again to 20 users at 600 seconds, causing another violation of the upper threshold as the available resources become inadequate for the increased traffic demand. *IntentContinuum* resolves this issue effectively within about 40 seconds by recommending more resources.The resolution time differs from that of the earlier, similar violation that occurred around the 100-second mark due to differences in system settings. Additionally, this violation resulted in a more severe impact, including longer queues and a greater degree

of performance degradation. At 720 seconds, when the user count decreases to 10, EMA_RT briefly dips below the lower threshold. Once more, *IntentContinuum* swiftly addresses the deviation, restoring the response time to the defined SLO within around 40 seconds.

Insight 1: *Results demonstrate that* IntentContinuum*, leveraging LLM recommendations, can dynamically adapt to varying load levels while maintaining the application's response time within the defined thresholds.*

To resolve these violations, *IntentContinuum* employs a range of actions, including adjusting pod placement, scaling replicas up or down, and dynamically reallocating resources such as CPU and memory. This demonstrates the system's ability to apply a variety of strategies—or a combination of them—to address violations effectively. In the following we discuss, the detailed actions recommended by *IntentContinuum*.

Figure 6(a) shows how the number (horizontal scaling) and size (vertical scaling) of replicas were adjusted during the experiment. Multiple lines represent horizontal scaling, while line thickness indicates CPU allocation per replica (0.5 cores for the thickest to 0.2 cores for the thinnest). Pod1, handling incoming traffic, and pod3, with heavier processing, scale dynamically, while pods 2 and 4 remain stable.

Table III illustrates how pod placement changes during the experiment due to detected violations and the corresponding corrective actions. It represents different pod placements: *init* denotes the initial placement, while V1, V2, V3, and V4 correspond to the system's responses to the first, second, third, and fourth violations, respectively. The table also indicates how pods are deployed on each node at each stage. The system recommends relocating pod3 in response to the first (V1) and third (V3) violations, while the other pods remain on their initial placement throughout the experiment.

Satate	Worker1	Worker2	Worker3
init	pod1	pod2	pod3, pod4
1st violation (V1)	pod1	pod2, pod3	pod4
2nd violation (V2)	pod1	pod2, pod3	pod4
3rd violation (V3)	pod1	pod2	pod3, pod4
4th violation (V4)	pod1	pod2	pod3, pod4

TABLE III: Pod Placement State

Insight 2: IntentContinuum *effectively combines vertical and horizontal scaling with pod replacements to optimize performance, ensure stability, and maintain capacity constraints.*

Comparison to Kubernetes Autoscaler: We also conducted experiments using the Kubernetes Horizontal Pod Autoscaler (HPA)[3] to demonstrate how our *IntentContinuum* outperforms the default scheduler in maintaining intent satisfaction. To the best of our knowledge, there is no comprehensive method that considers all aspects of the *IntentContinuum* together, making this comparison particularly valuable. The application was deployed with the same configuration described in the previous sections to ensure consistency and comparability. For the default scheduler, we configured a minimum of 1 replica for all pods and allowed scaling out to a maximum of 5 replicas as required. The CPU utilization thresholds for the autoscaler were set to 70%, 60%, and 50% across three separate experiments. Note that identifying the best threshold for HPA to maintain application response time within a specific range can be application specific and challenging in practice. Thus, we varied thresholds to enable us to evaluate the autoscaler's behavior in managing intent violations under different resource utilization conditions.

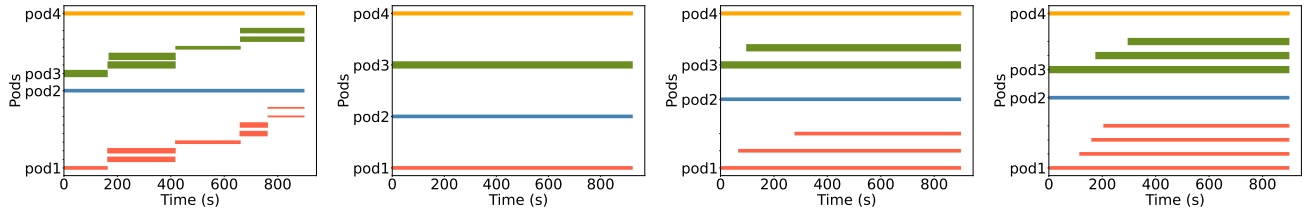
As shown in Figure 6(b), when the CPU threshold was set to 70%, the autoscaler did not trigger any action to increase the number of replicas, even under increased loads. This was because the average CPU utilization never exceeded the threshold. However, when the CPU utilization threshold was reduced to 60% and 50%, Figure 6(c) and Figure 6(d) the autoscaler responded to resource demands by scaling up the number of replicas. Specifically, it increased the number of replicas for pod1 and pod3 which were under pressure based on the CPU threshold.

Insight 3: *The adaptive nature of* IntentContinuum *eliminates the complexity of threshold setting in the application autoscaling process by focusing on high-level intents.*

Figure 7 illustrates the effect of varying autoscaler thresholds on application response time, with *IntentContinuum* included for comparison. At a threshold of 70%, the autoscaler does not scale up replicas, resulting in increased response times. In contrast, a threshold of 50% often leads to an overprovisioning of replicas, as the number remains higher than necessary for much of the time. However, a threshold of 60% strikes a balance by improving response time management and reducing resource utilization. Despite this improvement, the autoscaler is limited by its cooldown period, set to 5 minutes after detecting average CPU utilization, which prevents timely scaling down when the load decreases. Moreover, the autoscaler falls short in addressing violations as efficiently as *IntentContinuum* and struggles to manage upper and lower thresholds simultaneously with equal effectiveness, ultimately restricting precise control over resource scaling. Table IV presents the intent satisfaction rates and the total time during which the intent was violated for *IntentContinuum* and various configurations of the autoscaler. Based on the traffic patterns directed to the application, *IntentContinuum* consistently achieves a higher intent satisfaction rate compared to all autoscaler configurations. Specifically, *IntentContinuum* attains a satisfaction rate of approximately 85%, outperforming the autoscaler thresholds of 70%, 60%, and 50%, which achieve rates of 43%, 79.5%, and 82.5%, respectively. This is also show in the time spent in violation compared to the autoscaler.

Resource Usage: As shown in Table V, which reports normalized CPU and memory usage across all pods, *IntentContinuum* achieves the best resource utilization while adhering to the SLO defined in the intent. Similarly, Figure 8 illustrates the combined normalized resource usage across the different methods. While the autoscaler configured with a 70% threshold demonstrates the lowest CPU and memory usage, this comes

[3]https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/



(a) IntentContinuum (b) CPU threshold- 70% (c) CPU threshold- 60% (d) CPU threshold- 50%

Fig. 6: Pods Lifespans and The Number of Replicas Across Methods- Line Thickness Represents Vertical Scaling, While the Number of Lines Indicates Horizontal Scaling

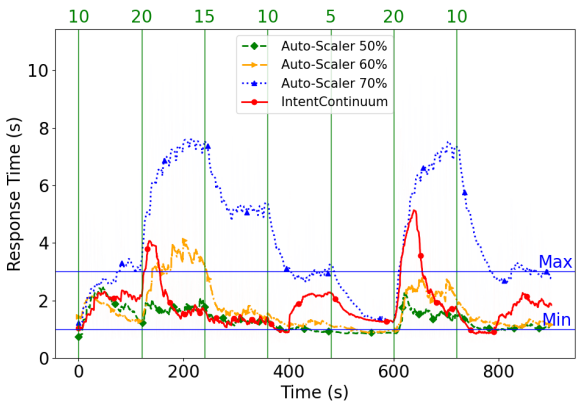


Fig. 7: Exponential moving average of the response time (EMA_RT) Across Methods

Metrics	*IntentContinuum*	Autoscaler		
		70%	60%	50%
Intent Satisfaction%	**85%**	43%	79.5%	82.5%
Total Amount of Violated Time (s)	**143**	509	184	157

TABLE IV: Comparison of Metrics Across Methods

at the cost of significantly higher application response times and the highest rate of intent violations. As the autoscaler threshold decreases, the system demonstrates increased resource usage and intent satisfaction, with the highest resource usage observed at a 50% threshold. This is when the closest satisfaction rate to that of the *IntentContinuum* is achieved, but this approach requires approximately 60% more CPU and memory usage.

Insight 4: IntentContinuum *provides the best balance between intent satisfaction and resource utilization compared to various Kubernetes autoscaler settings under varying traffic loads.*

Methods	Normalized Resource	Pods			
		P1	P2	P3	P4
Intent Continuum	CPU (core)	0.515	0.3	0.665	0.3
	Mem (MiB)	530.46	312	677.195	312
70%	CPU (core)	0.3	0.3	0.5	0.3
	Mem (MiB)	312	312	512	312
60%	CPU (core)	0.79	0.3	0.95	0.3
	Mem (MiB)	818	312	970	312
50%	CPU (core)	1.04	0.3	1.24	0.3
	Mem (MiB)	1083	312	1270	312

TABLE V: Normalized Resource Utilization Across Methods

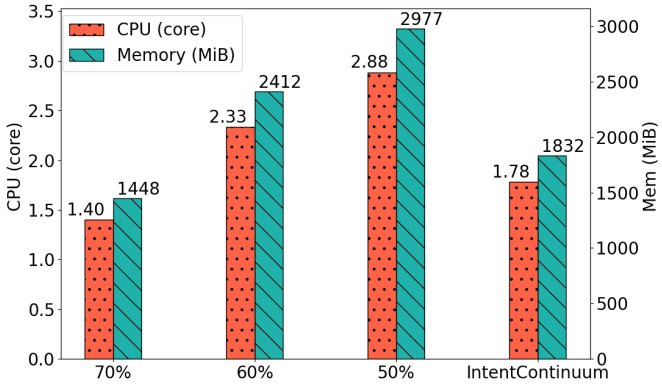


Fig. 8: Total Normalized Resource Usage Across Methods

*2) **Networking Experiment**:* Networking issues, such as link congestion, node failures, and link failures, can significantly affect application performance, making their consideration essential for robust intent management. Existing solutions in the literature often prioritize computing parameters while giving less emphasis to the combined impact of computing and networking factors. In this section, we perform an experiment to illustrate that *IntentContinuum* effectively addresses intent violations, even those caused by networking issues.

Locust is used to send HTTP requests to the application, with 10 users generating traffic at a spawn rate of 1 user per second over a duration of 900 seconds. Each request includes an image, and the generated traffic passes through the network switches as the pods are hosted on nodes interconnected through these switches. An initial traffic path is established between the pods, routed through the switches. Specifically, the initial route was configured as *S2-S4-S3-S2*, determined by the placement of the pods on the respective nodes. The image size remained consistent with previous experiments, but unlike before, the user load in Locust is kept constant throughout the test because the focus is specifically on networking issues, such as link congestion.

Figure 9 illustrates the response time observed during this experiment. At $e1$, congestion occurs on the link between switches *S2-S3*, introduced using *iPerf*[4] with the help of two external hosts. As shown in the figure, a violation is detected after a delay (marked as v in the figure). *IntentContinuum* effectively mitigates the issue by recommending an alternative route to bypass the congested link. Initially, traffic was routed through *S2-S4-S3-S2*; however, to address

[4]https://iperf.fr/

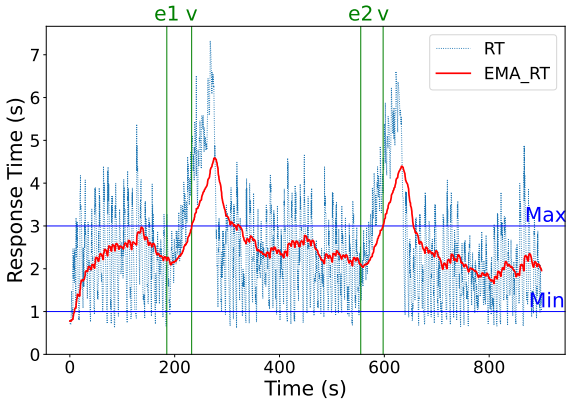
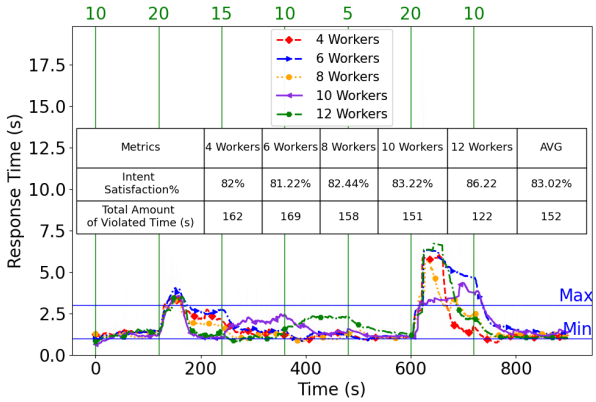


Fig. 9: Response Time for Networking Experiment



Metrics	4 Workers	6 Workers	8 Workers	10 Workers	12 Workers	AVG
Intent Satisfaction%	82%	81.22%	82.44%	83.22%	86.22	83.02%
Total Amount of Violated Time (s)	162	169	158	151	122	152

Fig. 10: EMA_RT Across Senarios

the congestion, the system implemented the recommended route via an API call to the SDN controller, which successfully updated the network flow. The new route utilized switches *S2-S4-S3-S1-S2*, ensuring efficient traffic flow and resolving the violation. Later in the experiment, at $e2$, another instance of *iPerf* was run simultaneously on two links, causing congestion between switches *S1-S2* and *S3-S4*. This congestion led to another violation, prompting *IntentContinuum* to send a request to the LLM asking for recommendations. The LLM suggested replacing pod3 from worker2 to worker1 to avoid the congested links. This action was carried out through an API call to the orchestrator, resulting in a new route being recommended: *S2-S4-S2*.

Insight 5: IntentContinuum *can address network issues such as link congestion or link failures by dynamically implementing flow scheduling or pod replacements through the combined use of an SDN controller and orchestrator.*

Insight 6: IntentContinuum *effectively utilizes networking and computing metrics, either independently or in combination, to address intent violations.*

C. Scalability Evaluation

In this section, we designed multiple scenarios with varying numbers of worker nodes to analyze the system's response to changes in node count. Given the constraints of our lab environment and the use of real cloud resources, we conducted experiments with 4, 6, 8, 10, and 12 worker nodes to evaluate the feasibility of *IntentContinuum* under different conditions. Figure 10 presents the EMA_RT across these scenarios, using the same traffic load as in Figure 5. The results indicate that the intent satisfaction rate remains nearly consistent across different methods, with an average satisfaction rate of 83.02%. To further evaluate scalability, we simulated scenarios with a larger number of nodes and sent the corresponding prompts to GPT-4o, measuring the latency—the time taken from when a prompt is sent to GPT-4o until a response is received. In each scenario, we introduced five violations and recorded both the average latency and the average token count sent to GPT-4o (tokens in) and received from it (tokens out) across these violations. This analysis illustrates how the system adapts as the number of nodes increases. The results are summarized in Table VI. Notably, GPT-4o's latency remained relatively stable across different scenarios, despite variations in the number of tokens processed. However, at 700 nodes, it failed to generate responses for some prompts due to token limitations, marking an upper boundary for direct scalability. Nevertheless, *IntentContinuum* can accommodate larger node counts by partitioning complex prompts into smaller, manageable segments, ensuring continued scalability. Further optimizations remain an area for future work.

# of Nodes	10	50	100	200	300	400	500	600
Latency (s)	13.37	13.95	13.80	14.50	12.80	12.69	12.45	13.42
# of tokens in	4916	7196	10044	15745	21445	27145	32846	38552
# of tokens out	570	522	736	561	446	498	544	525

TABLE VI: Average GPT-4o Latency and Token Usage Across Different Node Scenarios over Five injected Violations

V. Discussions

A. Strengths

Our *IntentContinuum* demonstrates several key strengths. It optimizes CPU and memory usage, efficiently manages replicas, and ensures that predefined intents and SLOs are consistently met within acceptable limits. By balancing resource allocation, it minimizes waste while adhering to defined thresholds. Additionally, *IntentContinuum* effectively manages both networking and computing parameters, demonstrating its versatility in handling diverse features. Note that while the framework leverages GPT-4o for decision-making, its design supports seamless integration with other LLMs. Moreover, LLM serves as auxiliary add-on features to the system, ensuring that even if it becomes unavailable, the application remains operational, albeit with potential impacts on QoS. In such cases, the framework can revert to built-in mechanisms for scheduling, autoscaling, and network flow management provided by tools like Kubernetes and the ONOS controller. This ensures effective handling of scenarios such as scaling, node failures, and network switch or link issues. This design enhances the robustness and resilience of the framework, ensuring it remains functional despite potential disruptions.

B. Limitations

Despite its strengths, *IntentContinuum* has certain limitations that must be addressed to enhance its adaptability to a broader range of use cases and operational environments:

Dependence on models: The framework relies on GPT-4o for its decision-making. While this helps to improve automation, it may not always provide the best solutions in complex or unusual scenarios. The quality and accuracy of decisions also depend on the data fed into the model, which could affect performance in cases of incomplete or noisy data.

Limited Transparency and Clarity of LLM Recommendations: While GPT-4o can provide real-time resource management decisions, the reasoning behind some of its decisions may not be fully transparent. This black-box nature of LLMs could make it difficult to understand why certain actions were recommended, which may limit our *IntentContinuum* framework's adoption in environments where transparency and clarity are critical.

Scalability and Processing Overhead: Context Length for LLM models such as GPT-4o is limited. As the system scales up, the computational demands on GPT-4o could also introduce delays, particularly when processing large datasets in real time. In this paper, we focused on evaluating whether LLMs can provide effective recommendations for resource management in application deployments across the compute continuum. However, further research is required to explore their full scalability potential and identify possible performance bottlenecks at larger scales, which we leave as part of our future work.

Financial implications: As data volumes grow, the number of tokens exchanged with the LLM increases, resulting in higher costs. This impact becomes more pronounced as the application scales, incorporating a larger number of edge nodes, cloud nodes, IoT devices, and microservices.

VI. Related Work

Research on resource management in the compute continuum has explored various techniques for optimizing task allocation and system performance. These techniques range from traditional algorithms to emerging AI-driven methods, each with unique strengths and limitations.

a) Traditional Resource Management Approaches: Traditional resource allocation methods, such as heuristic and meta-heuristic algorithms, are widely used for their ability to provide near-optimal solutions efficiently [9]. Techniques like genetic algorithms (GA) [10], particle swarm optimization (PSO) [18], and simulated annealing [11] excel in static or smaller systems but struggle with the scalability and real-time adaptability required in dynamic compute continuum environments [19]. While advancements like dynamic parameter tuning in GA [20] improve flexibility, these methods often rely on manual adjustments, limiting their effectiveness in unpredictable, distributed systems [21].

b) Edge-Cloud Resource Management and AI-Driven Intents: Resource allocation across edge and cloud infrastructures remains a critical research area, with strategies like task offloading reducing latency and energy consumption [22]. However, static policies often fail under rapidly changing workloads, prompting the adoption of dynamic workload partitioning [23]. While these advances improve adaptability, the real-time responsiveness required for complex IoT systems remains challenging, as current methods struggle to continuously monitor and respond to environmental changes.

AI techniques, including reinforcement learning (RL) and predictive deep learning models, have enhanced resource management by enabling automated adjustments and anticipating resource needs [24]. RL has been effective in optimizing specific parameters like latency [25] but requires extensive training data, limiting its applicability to dynamic environments [26]. Predictive models improve efficiency using historical data but often fail to adapt to unforeseen changes [27].

Intent-driven systems offer a promising approach, translating user-defined goals into automated policies for resource management [28]. While these systems provide greater flexibility than traditional methods, their reliance on static intent mappings limits their adaptability to the dynamic nature of edge-cloud environments [29]. Future systems must incorporate more dynamic, context-aware mechanisms to address the challenges of distributed, heterogeneous environments.

c) Large Language Models in System Management: Large language models like ChatGPT and Llama are transforming system management in distributed environments such as the compute continuum. With their ability to process natural language, these models automate decision-making, detect anomalies, predict resource bottlenecks, and optimize scheduling and resource allocation [30], [31]. Their adaptability to changing conditions enables proactive, intent-driven management, reducing human intervention and improving operational efficiency.

VII. Conclusions and Future Work

In this paper, we presented a comprehensive framework for intent-driven resource management in the compute continuum, leveraging the capabilities of LLMs to address the complexities associated with deploying applications across edge and cloud environments. The proposed *IntentContinuum* framework effectively ensures that user-defined intents, such as maintaining application response times within a specified range, are consistently satisfied. By incorporating real-time system monitoring, root cause analysis, and adaptive corrective actions, *IntentContinuum* significantly reduces the need for human intervention and simplifies the complexity of resource management. Furthermore, *IntentContinuum* demonstrates the ability to manage both networking and computing parameters simultaneously, ensuring seamless operation and improved system reliability. Our approach outperforms traditional methods by maintaining system stability under varying workloads, optimizing resource utilization, and dynamically adapting to changing conditions. The findings from our real-world proof-of-concept implementation and experimental evaluation underscore the effectiveness of our proposed method. In future work we aim to enhance the transparency of the framework, reduce computational overhead, and further improve its cost-effectiveness and scalability, enabling broader adoption and applicability in real-world scenarios.

ACKNOWLEDGEMENTS

Akbari is supported by a Faculty of IT PhD scholarship. Grundy is supported by ARC Laureate Fellowship FL190100035.

REFERENCES

[1] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] H. Tabrizchi and M. Kuchaki Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The journal of supercomputing*, vol. 76, no. 12, pp. 9493–9532, 2020.

[3] G. R. Russo, V. Cardellini, and F. L. Presti, "Serverless functions in the cloud-edge continuum: Challenges and opportunities," in *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2023, pp. 321–328.

[4] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE Access*, vol. 8, pp. 85 714–85 728, 2020.

[5] M. Sajid and Z. Raza, "Cloud computing: Issues & challenges," in *International conference on cloud, big data and trust*, vol. 20, no. 13. sn, 2013, pp. 13–15.

[6] M. Danelutto, P. Dazzi, and M. Torquati, "Structuring the continuum," in *International Conference on Advanced Information Networking and Applications*. Springer, 2024, pp. 212–223.

[7] M. Guzek, P. Bouvry, and E.-G. Talbi, "A survey of evolutionary computation for resource management of processing in cloud computing," *IEEE Computational Intelligence Magazine*, vol. 10, no. 2, pp. 53–67, 2015.

[8] A. K. Sangaiah, A. A. R. Hosseinabadi, M. B. Shareh, S. Y. Bozorgi Rad, A. Zolfagharian, and N. Chilamkurti, "Iot resource allocation and optimization based on heuristic algorithm," *Sensors*, vol. 20, no. 2, p. 539, 2020.

[9] V. Sharma and A. K. Tripathi, "A systematic review of meta-heuristic algorithms in iot based application," *Array*, vol. 14, p. 100164, 2022.

[10] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[11] T. Guilmeau, E. Chouzenoux, and V. Elvira, "Simulated annealing: A review and a new scheme," in *2021 IEEE statistical signal processing workshop (SSP)*. IEEE, 2021, pp. 101–105.

[12] V. Millnert and J. Eker, "Holoscale: Horizontal and vertical scaling of cloud resources," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2020, pp. 196–205.

[13] A. Marchese and O. Tomarchio, "Network-aware container placement in cloud-edge kubernetes clusters," in *2022 22nd IEEE international symposium on cluster, cloud and internet computing (CCGrid)*. IEEE, 2022, pp. 859–865.

[14] A. N. Toosi, J. Son, Q. Chi, and R. Buyya, "Elasticsfc: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *Journal of Systems and Software*, vol. 152, pp. 108–119, 2019.

[15] N. Akbari, A. N. Toosi, J. Grundy, H. Khalajzadeh, M. S. Aslanpour, and S. Ilager, "iContinuum: An emulation toolkit for intent-based computing across the edge-to-cloud continuum," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 2024, pp. 468–474.

[16] N. Fareghzadeh, M. A. Seyyedi, and M. Mohsenzadeh, "Dynamic performance isolation management for cloud computing services," *The Journal of Supercomputing*, vol. 74, pp. 417–455, 2018.

[17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[18] T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, "Particle swarm optimization: A comprehensive survey," *Ieee Access*, vol. 10, pp. 10 031–10 061, 2022.

[19] F. Xhafa and A. Abraham, *Metaheuristics for scheduling in distributed computing environments*. Springer, 2008, vol. 146.

[20] H. Materwala, L. Ismail, and H. S. Hassanein, "Qos-sla-aware adaptive genetic algorithm for multi-request offloading in integrated edge-cloud computing in internet of vehicles," *Vehicular Communications*, vol. 43, p. 100654, 2023.

[21] S. Gupta and N. Singh, "Heuristics and meta-heuristics based algorithms for resource optimization in fog computing environment: A comparative study," in *2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, 2023, pp. 271–276.

[22] F. Liu, Z. Huang, and L. Wang, "Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for iot sensors," *Sensors*, vol. 19, no. 5, p. 1105, 2019.

[23] Z. Cao, B. Xiao, H. Duan, L. Yang, and W. Cai, "A dynamic partitioning framework for edge-assisted cloud computing," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2020, pp. 215–229.

[24] T. Zheng, J. Wan, J. Zhang, and C. Jiang, "Deep reinforcement learning-based workload scheduling for edge computing," *Journal of Cloud Computing*, vol. 11, no. 1, p. 3, 2022.

[25] M. Khani, M. M. Sadr, and S. Jamali, "Deep reinforcement learning-based resource allocation in multi-access edge computing," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 15, p. e7995, 2024.

[26] S. Wang, Y. Li, and F. Chen, "Optimizing blue team strategies with reinforcement learning for enhanced ransomware defense simulations," 2024.

[27] J. Hurtado, D. Salvati, R. Semola, M. Bosio, and V. Lomonaco, "Continual learning for predictive maintenance: Overview and challenges," *Intelligent Systems with Applications*, vol. 19, p. 200251, 2023.

[28] M. Kyryk, N. Pleskanka, M. Pleskanka, and V. Kyryk, "Infrastructure as code and microservices for intent-based cloud networking," in *Future Intent-Based Networking: On the QoS Robust and Energy Efficient Heterogeneous Software Defined Networks*. Springer, 2021, pp. 51–68.

[29] C. Sicari, A. Catalfamo, L. Carnevale, A. Galletta, A. Celesti, M. Fazio, and M. Villari, "Toward the edge-cloud continuum through the serverless workflows," in *Device-Edge-Cloud Continuum: Paradigms, Architectures and Applications*. Springer, 2023, pp. 1–18.

[30] T. Mongaillard, S. Lasaulce, O. Hicheur, C. Zhang, L. Bariah, V. S. Varma, H. Zou, Q. Zhao, and M. Debbah, "Large language models for power scheduling: A user-centric approach," *arXiv preprint arXiv:2407.00476*, 2024.

[31] Z. Ji, J. Zhang, and X. Wang, "Intelligent scheduling strategies for computing power resources in heterogeneous edge networks," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 2022, pp. 253–271.