CognitiveContinuum: Intent-Aware Orchestration Using LLM and Fuzzy Logic

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

***Abstract*—The heterogeneity of compute continuum environments challenges traditional orchestration frameworks that depend on low-level directives and rigid rules. Users, however, think in terms of goals such as responsiveness, cost-efficiency, energy savings, or reliability rather than infrastructure details. To bridge this gap, we present *CognitiveContinuum*, an intent-aware orchestration framework integrating three modules: large language models (LLMs) to interpret natural language intents, a fuzzy reasoning engine for adaptive decision-making, and a Greedy Scheduler (GS) extending Kubernetes to optimize replica placement across edge and cloud nodes. *CognitiveContinuum* translates natural language and high-level user objectives into provisioning and placement strategies, dynamically rebalancing configurations as workloads evolve. Built on Kubernetes and SDN controllers, it integrates seamlessly with existing platforms. Evaluation on a multi-service IoT application shows 95% accuracy in intent interpretation and consistent fulfillment of user objectives, balancing latency, cost, power, and reliability across varying workloads.**

Keywords- Intent-driven orchestration, Edge–cloud continuum, Large Language Models (LLMs), Fuzzy logic decision-making, Adaptive resource management.

I. Introduction

Modern distributed applications span a continuum of resources, from edge devices to cloud servers, to support latency-sensitive and dynamic workloads such as smart surveillance, real-time analytics, and industrial IoT [1], [2]. While this model improves performance and adaptability, it increases orchestration complexity [3] and requires balancing latency, energy use, reliability, and cost [4]. Current orchestration systems mainly rely on threshold-based autoscaling and metric-driven policies [5], which optimize metrics independently rather than considering their interdependencies [6], [7], depend on expert tuning, assume stable workloads, provide limited flexibility for user intent [3], and often fail to capture trade-offs between objectives or adapt to changing system conditions. Recent research has explored multi-objective orchestration [8], SLO-based management, and intent-driven frameworks [9] that aim to incorporate higher-level goals into deployment decisions, but most existing approaches rely on predefined objective mappings, optimization heuristics, or learning-based policies, and do not explicitly integrate natural language intent interpretation with structured multi-objective reasoning and practical enforcement within a unified architectural pipeline.

A key limitation is an orchestration model that aligns system behavior with human goals [10]. Users think in terms of intent, like wanting faster response time or reliable operation with low energy use, not replicas or CPU quotas. These goals are often vague and involve trade-offs, which current systems cannot interpret or manage effectively.

Imagine a user deploying an IoT application across edge and cloud resources. Instead of specifying detailed configurations, the user expresses high-level goals: *fast responsiveness for time-critical tasks, tolerance for moderate resource usage, minimal energy consumption, and strong dependability for safety-critical components.* Such objectives are natural for humans but lack precision. Conventional orchestration tools cannot reason over these abstract goals, relying instead on fixed rules and thresholds that assume expert input. In contrast, users typically think in terms of intents and tolerances, not system parameters. This highlights the need for orchestration that translates general user expectations into concrete deployment actions. Thus, we aim to address the following key research question: *"How can we translate users' natural-language intents into system objectives, balance trade-offs among those objectives, and adapt decisions as workloads change?"*

To address this, we propose an intent-aware orchestration framework, *CognitiveContinuum*, combining Large Language Models (LLMs), fuzzy logic [11], and a Greedy Scheduler (GS). The LLM interprets natural language intents, the fuzzy engine resolves ambiguities and trade-offs [12] to generate concrete actions, and the GS enforces them by guiding microservice (pod, a lightweight deployable unit in Kubernetes) placement across cloud and edge nodes based on cost and reliability. This hybrid design enables adaptive orchestration that understands user goals and adjusts deployments in real time, scaling toward the edge for latency-sensitive tasks or to the cloud when cost and energy allow. By unifying user intent, fuzzy reasoning, and adaptive deployment, this work moves toward autonomous, user-aligned resource management across edge and cloud environments. Our **key contributions** in this work include: 1) a unified framework that captures natural language intent and converts it into deployment logic using LLMs and fuzzy reasoning; 2) an adaptive orchestration mechanism that balances multiple objectives without requiring user-level expertise in configuration; and 3) a practical demonstration of the proposed orchestration across the compute continuum, driven solely by abstract user goals.

II. Our Approach

Figure 1 shows the *CognitiveContinuum* framework for deploying applications across the compute continuum. Natural-language user intents are translated into system-level decisions via LLMs and fuzzy logic. Key modules, including *Intent Interpretation*, are detailed below; source code is on GitHub.[1]

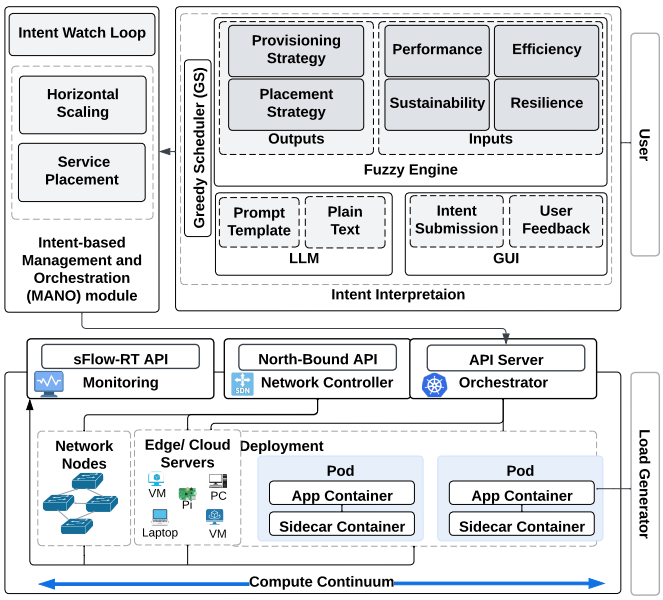


Fig. 1: Architectural framework of our *CognitiveContinuum*

Compute Continuum: Extending the *Compute Continuum* model from iContinuum [13], Kubernetes orchestrates edge/-cloud nodes running microservices as pods, with nodes connected through SDN-managed switches. All components are continuously monitored, with requests entering the ingress gateway and user-defined intents guiding application behavior.

Intent Interpretation: *Intent Interpretation* is *CognitiveContinuum*'s component, operating within the continuum infrastructure. It comprises several components, detailed below.

1) ***User Interface (GUI)****:* The *GUI* is the main interface for submitting natural-language requirements through *Intent Submission*, such as reducing latency, energy, or cost, or improving reliability, without technical settings. Inputs are sent to the LLM for formalization, and the GUI provides text and visual feedback on fulfillment.

2) ***Large Language Model (LLM)****:* The *LLM* module uses GPT-4o with zero-shot prompting [14] to interpret intent and assign high/medium/low priorities to response time, cost, power, and reliability, outputting JSON for fuzzy decisions (Figure 2). The framework is model-agnostic, but GPT-4o was chosen for reliable zero-shot reasoning and stable structured output; since it runs only at intent submission, it adds no runtime latency and negligible infrastructure cost [3].

3) ***Fuzzy Engine****:* The fuzzy engine is the framework's core decision unit and operates independently of the LLM. The LLM converts unstructured intent into objective priorities,

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

```
You are an IoT assistant. Given a user's intent, rate response_time, cost, power, and reliability
as high, medium, or low based on the importance level. Follow reasoning rules on constraints,
feasibility, tolerance, and frequency. Default to medium if unspecified. Return the result in the
following JSON format:
 {
"intent": {
   "response_time": "<low| medium|high>"
   "cost": "<low| medium|high>",
   "power": "<low| medium|high>",
   "reliability": "<low| medium|high>"
 }}
```

Fig. 2: GPT-4o prompt for translating intent to metrics.

while the fuzzy engine performs quantitative multi-objective reasoning on real-time metrics. Unlike fixed intent-action mappings, it handles gradual metric changes, conflicting objectives, and partial priorities through continuous membership functions, enabling smoother adaptation under dynamic workloads. This layered design separates language understanding from control logic, improving transparency, modularity, and stability. The engine takes four normalized inputs in [0, 1], *performance* (response time), *efficiency* (cost), *sustainability* (power), and *resilience* (reliability), and maps them to two outputs in [0, 1], *provisioning* and *placement*, using 81 expert-defined fuzzy rules for edge/cloud trade-offs. *Provisioning* sets pod replicas per microservice, and *placement* sets deployment preference (low = edge, high = cloud). The full rule set is in the same GitHub repository, with an example below:

```
IF response_time=low, cost=low, power=low, reliability=low
THEN provisioning_strategy=few, placement_strategy=low
```

Here, no objective is strongly prioritized, so the engine recommends few replicas and edge-leaning deployment to conserve resources. For example, a placement of 0.3 allocates 70% of replicas to edge and 30% to cloud, keeping decisions aligned with user intent across heterogeneous environments.

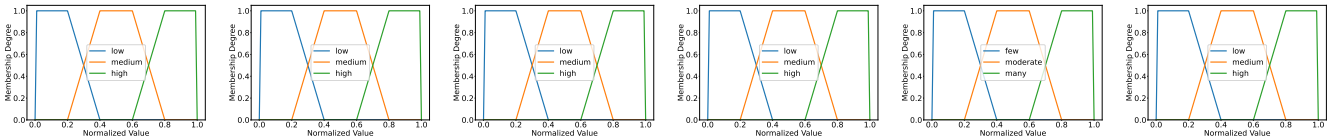
4) ***Greedy Scheduler (GS)****:* To enforce fuzzy decisions, *GS* adds a lightweight layer above Kubernetes that converts normalized outputs into actions: provisioning sets replica counts within traffic-based bounds, and placement sets the edge/cloud split. For each microservice, it creates `-edge`/`-cloud` deployments with node-group selectors, ranks nodes (cloud by lower cost, edge by lower failure probability), and applies soft affinity through *preferredDuringSchedulingIgnoredDuringExecution*[2], while the default scheduler performs final binding. Because trade-offs are already resolved by the fuzzy engine, this greedy design remains simple and Kubernetes-compatible.

A. Detailed overview of the Fuzzy Engine

1) ***Fuzzy Inputs****:* The Fuzzy Engine represents each input with trapezoidal membership functions at three importance levels (***low***, ***medium***, ***high***). Normalized inputs are evaluated by a Mamdani FIS[3] using minimum-based AND for rule antecedents. Membership parameters are in our GitHub repository, and related functions are shown in Figure 3 (3(a), 3(b), 3(c), 3(d)); each input's computation and role are described below.

[2]https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

[3]https://www.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html



(a) Response time (b) Cost (c) Power (d) Reliability (e) Provisioning Strategy (f) Placement Strategy

Fig. 3: Fuzzy membership functions

Performance: Performance is captured by the application's end-to-end response time, smoothed via the *Exponential Moving Average (EMA)* [3]:

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

where RT_t is the latest response time and α is the smoothing factor (e.g., $\alpha = 0.02$). This emphasizes recent samples while preserving history, enabling proactive adaptation under fast performance intent.

Efficiency: Efficiency is measured by *Cost_Rate*, reflecting CPU and memory usage on hosting nodes:

$$\text{Cost\_Rate}_n = \sum_{p \in \mathscr{P}_n} \left(\frac{\text{CPU}_p}{\text{CPU}_n^{\text{total}}} \times c_{\text{cpu}} + \frac{\text{MEM}_p}{\text{MEM}_n^{\text{total}}} \times c_{\text{mem}} \right) \quad (2)$$

where CPU_p, MEM_p are pod p's CPU and memory usage, $\text{CPU}_\text{n}^\text{total}$, $\text{MEM}_\text{n}^\text{total}$ are node n's total CPU and memory, and c_{cpu}, c_{mem} represent the cost per unit CPU and memory. We use AWS on-demand `t2.micro` EC2 instance pricing[4]: \$0.0058 per vCPU/hour and \$0.000005664 per MiB/hour, and assume edge resources are about 80% cheaper than cloud nodes [15].

Sustainability: Sustainability is captured via power consumption, driven by node CPU usage:

$$P_{\text{node}} = P_{\text{idle}} + (P_{\text{max}} - P_{\text{idle}}) \times \left(\frac{\text{CPU}_{\text{used}}}{\text{CPU}_{\text{total}}} \right) \quad (3)$$

where P_node is the node's total power; P_idle and P_max denote idle and full-load power; CPU_used is the total CPU used by pods; and CPU_total is the node's total CPU capacity. P_idle and P_max vary for edge and cloud nodes. For cloud nodes, we use `t4g.xlarge` EC2 reference values (6.2 W idle, 21 W full load) from a estimator;[5] edge nodes are assumed to consume 20% less [16]. The application power on the node is:

$$P_{\text{cons}} = P_{\text{node}} \times \left(\frac{\sum_{i=1}^{N} \text{CPU}_i}{\text{CPU}_{\text{total}}} \right) \quad (4)$$

where P_cons is the total power consumed by all pods on the node, CPU_i is the CPU usage of pod i, and N is the number of pods on the node.

Resilience: Resilience captures application reliability via node failure probabilities. Cloud nodes are highly reliable (less than 0.1%) [17], while edge nodes fail more often due to resource limits and decentralization. Each node i is assigned a failure probability $\text{P}_\text{fail,i}$, randomly sampled from a predefined range based on node type: cloud nodes use [0.001, 0.005] (more reliable) and edge nodes use [0.01, 0.10] (less reliable). Each pod is deployed on a specific node and inherits the failure probability of that node, $\text{P}_\text{fail,p} = \text{P}_\text{fail,n(p)}$, where n(p) denotes the node hosting pod p. A microservice with r replicas is considered operational as long as at least one of its pods is running. Assuming independent failures, the reliability of a microservice is:

$$R_{\text{microservice}} = 1 - \prod_{j=1}^{r} P_{\text{fail},j} \quad (5)$$

where $\text{P}_\text{fail,j}$ is the failure probability of the node hosting replica j. The product term represents the probability of all replicas failing simultaneously.

The application consists of m microservices connected sequentially, e.g., microservice 1 $\rightarrow$ microservice 2 $\rightarrow$... $\rightarrow$ microservice m. The application is successful only if all microservices are operational:

$$R_{\text{application}} = \prod_{k=1}^{m} R_{\text{microservice},k} \quad (6)$$

where $\text{R}_\text{microservice,k}$ denotes the reliability of the k-th microservice. This reflects the fact that a failure in any microservice leads to application-level failure.

*2) **Fuzzy Outputs**:* The fuzzy engine produces two outputs, each with its own membership functions and defuzzified via the centroid method (Figure 3(e),3(f)).

Provisioning Strategy: The provisioning strategy sets replicas per microservice (*few*, *moderate*, *many*). To derive valid ranges, we profiled Kubernetes HPA[6] with Locust[7] traffic (1 to 40 users, ramp-up of 1 user/s) under two settings: 10% CPU on edge (faster scaling, lower response time, more replicas) and 20% CPU on cloud (slower scaling, higher response time, lower resource use). Thresholds below 10% were unstable, while higher thresholds violated response-time targets, defining lower and upper bounds for replicas and response time. On intent submission, the system reads Locust traffic, scales using Table I, and applies fuzzy rules to set each microservice's *Provisioning Strategy* and replica count. *CognitiveContinuum* computes the final number of replicas using the following formula:

$$R_i = \left\lfloor \mu_{\min,i} + \rho \times \left(\mu_{\max,i} - \mu_{\min,i} \right) \right\rceil, \quad i \in \{1, 2, \ldots, N\} \quad (7)$$

Where R_i is the replica count for microservice i, rounded to the nearest integer using $\lfloor \cdot \rceil$; $\rho \in [0,1]$ is the scaling factor from the fuzzy output (*provisioning strategy*); and $\mu_{\min}$, $\mu_{\max}$ are the minimum and maximum replicas for microservice i.

[4]https://aws.amazon.com/ec2/pricing/on-demand/

[5]https://engineering.teads.com/sustainability/carbon-footprint-estimator-for-aws-instances

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

[7]https://locust.io/

# Users	$\mu 1$		$\mu 2$		$\mu 3$		$\mu 4$		RT (s)	
	min	max	min	max	min	max	min	max	min	max
1	1	2	1	1	1	2	1	1	1.5	6.5
5	1	5	1	1	2	6	1	1	1.5	6.5
10	2	10	1	2	3	10	1	1	1.5	6.5
15	3	14	1	3	4	17	1	1	1.5	6.5
20	3	17	1	4	4	21	1	2	1.5	6.5
25	4	24	1	4	5	27	1	2	1.5	6.5
30	5	27	1	5	6	30	1	3	1.5	6.5
35	6	32	2	5	7	36	1	3	1.5	6.5
40	6	37	2	6	8	41	1	3	1.5	6.5

TABLE I: Replica allocation and RT of $\mu 1$–$\mu 4$ under loads.

Placement strategy: After determining total replicas R per microservice, the second fuzzy output, *placement strategy*, distributes them between cloud and edge nodes. It uses $\lambda \in [0,1]$, where $\lambda = 1$ means full cloud placement, and $\lambda = 0$ full edge placement. Cloud replicas are $\lfloor \lambda \times \mathrm{R} \rceil$, and the remainder go to edge nodes. For example, if $\mathrm{R} = 5$ and $\lambda = 0.4$, the cloud replica count is:

$$\lfloor 0.4 \times 5 \rceil = \lfloor 2.0 \rceil = 2$$

Thus, 2 replicas will be deployed on cloud nodes and the remaining 3 replicas on edge nodes.

$$R_{\text{cloud}} = \lfloor R \times \lambda \rceil, \quad R_{\text{edge}} = R - R_{\text{cloud}} \tag{8}$$

To enforce this split, GS labels nodes as `cloud` or `edge` and creates two deployments per microservice (`-cloud`, `-edge`) using `nodeSelector`. Soft `NodeAffinity` prefers cost-efficient cloud nodes and reliable edge nodes without violating constraints. Both deployments share one Kubernetes service for unified routing, and the original deployment is removed so execution matches user intent.

Intent-based Management and Orchestration (MANO): After the fuzzy engine produces provisioning and placement strategies, *MANO* enforces them via the *IntentWatchLoop* [3], aligning system state with user intent by placing pods and scaling replicas in Kubernetes per fuzzy outputs.

III. Performance Evaluation

A. Experimental Setup

To evaluate *CognitiveContinuum*, we use the three-layer topology in Figure 4 (**cloud**, **edge**, **core**); configured inter-layer bandwidth and delay are also shown there. Mininet[8] emulates the switches managed by ONOS[9] SDN controller. The cluster has one master and 20 workers (8 cloud, 12 edge), all on Ubuntu 20.04 LTS (amd64); the master provides 32 vCPUs/64 GB RAM, and each worker 4 vCPUs/8 GB RAM. Each server is modeled by cost, power consumption, and failure probability. Cost is based on vCPU and memory pricing measured in USD ($) (cloud: $0.0058 per vCPU-hour and $0.000005664 per MiB-hour; edge: 80% lower), with application Cost_Rate from Eq. 2. Power is derived from CPU usage and replica count, with node and application power computed by Eq. 3 and 4. Failure probability is used to compute application reliability via Eq. 6. Using these parameters, the system computes application cost, power, and reliability, while response time (calculated as Eq. 1) is read from the monitoring database to form baseline objective values. When a user submits an intent, the fuzzy-logic engine applies the matching rule and generates a balanced configuration, determining microservice placement (edge/cloud), replica distribution ratio, and total replicas per microservice.

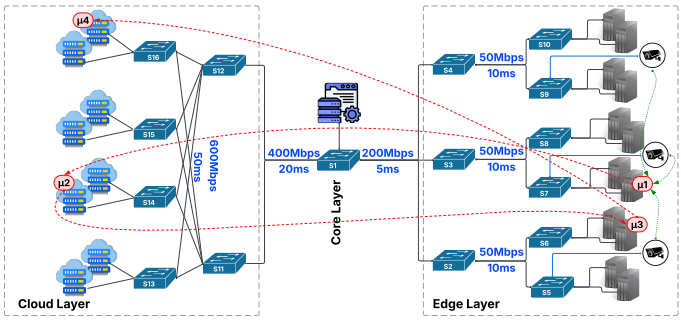


Fig. 4: Three-layer topology: cameras send traffic to μ_1; links show bandwidth (Mbps) and delay (ms).

We use a four-stage image-processing pipeline (μ_1, μ_2, μ_3, and μ_4) in Figure 4: μ_1 resizes images, μ_2 converts to grayscale, μ_3 performs object detection, and μ_4 triggers an alarm on target detection. The microservices run as Kubernetes pods, exposed through Service, and are initially placed by the default scheduler. IoT traffic is emulated with distributed Locust[10] using one controller and three CCTV-emulator nodes sending data to μ_1.

B. Results and Analysis

This section evaluates *CognitiveContinuum* under diverse scenarios and shows that it consistently satisfies user-defined intents through dynamic orchestration across the compute continuum, even as conditions change. Workloads were generated with Locust at 10, 20, 30, and 40 concurrent users, ramped at one user per second. Each load level was run independently for 1,420 seconds with constant traffic. In each run, five intents were submitted via the GUI Intent Submission module at predefined times to emulate runtime priority shifts. In this prototype, submission is externally triggered for controlled evaluation; in production, application events (for example, object detection) could trigger the same module through an API or event-driven interface. The test application is an image-processing pipeline relevant to home security, classroom monitoring, parental supervision, and public safety. Users submit intents prioritizing latency, reliability, cost, or energy efficiency. The five evaluated intents are listed below.

- Intent A (Home Security- Motion Detection): *When motion is detected by home security cameras, alerts should be generated quickly and with high reliability. The system should remain reasonably affordable and minimize energy*

[8]https://mininet.org/overview/

[9]https://opennetworking.org/onos/

[10]https://docs.locust.io/en/stable/running-distributed.html

consumption on battery-powered edge devices such as IoT cameras.

- Intent B (Smart Classroom- Lecture Capture): *Cameras should capture an image of the classroom whiteboard at the end of each lecture. Low latency is not required, and power is not constrained since electricity is always available. The system should be cost-efficient to allow deployment in every classroom.*

- Intent C (Parental Monitoring- Frequent Activity Alerts): *The system should notify parents when frequent motion is detected in a child's room. Reliability is critical to avoid missed events. Since cameras may be battery-powered, energy efficiency is important, while cost is less of a concern.*

- Intent D (Public Safety- Prohibited Item Detection): *Security cameras must instantly detect and raise alerts if prohibited objects are identified. Reliability is non-negotiable, whereas budget and energy usage are secondary priorities.*

- Intent E (Facility Monitoring- Classroom Coverage): *All classrooms should be monitored continuously at minimal cost. Higher delays are acceptable for alerts. Energy efficiency is desirable, especially when classrooms are unoccupied, while reliability and stability are less critical in this scenario.*

These intents are submitted through the Intent Submission Module to encode user requirements as metric priorities. Table II shows GPT-4o mappings from natural language to these priorities, with bold values indicating required objectives. The selected intents cover realistic IoT scenarios with different objective emphasis and, since fuzzy inference operates on continuous normalized inputs, the system generalizes beyond these representative cases.

Intent#	Response Time	Cost	Power	Reliability
A	**high**	medium	medium	**high**
B	low	**high**	medium	low
C	medium	low	**high**	**high**
D	**high**	low	low	**high**
E	low	**high**	medium	low

TABLE II: Intent-to-metric translation by GPT-4o

Response Time: Figure 5 shows application response time during each run, with time on the x-axis (1,420 s) and measured end-to-end response time (s) on the y-axis. The plotted results correspond to traffic levels 10, 20, 30, and 40, and green dashed lines A-E mark intent submission times. As shown in Table II, response time is a high-priority objective for intents A and D; after these submissions, latency decreases consistently across all traffic levels due to added microservice replicas. For A and D, most replicas are placed on cloud nodes to improve reliability, resulting in slightly higher latency than edge deployments. The pie charts above the figure show this trade-off, with cloud-dominant placement for A and D and edge-dominant placement for B and E.

Figure 6 shows replica counts for each microservice across traffic levels, with traffic level (10, 20, 30, 40) on the x-axis and replica count on the y-axis in each subplot. Colors indicate intents A-E. In the legend, labels such as "A-Before" and "A-After" represent replica counts before and after Intent A in the same run, with the same meaning for B-E. The largest increases occur in microservice1 and microservice3 after submitting intents A and D. As shown in Figure 6(a), replica counts rise proportionally with traffic, and the same scaling appears in Figures 6(b)–6(d)

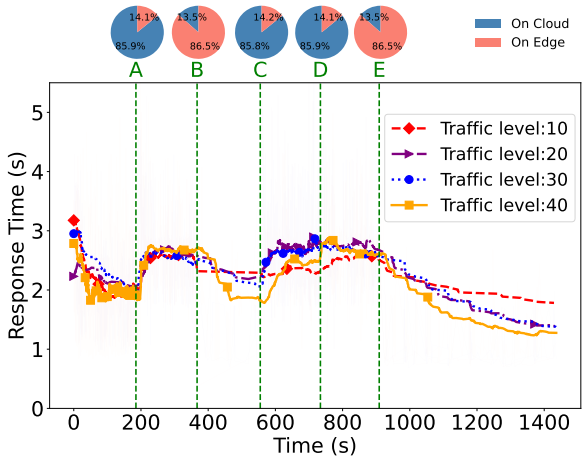


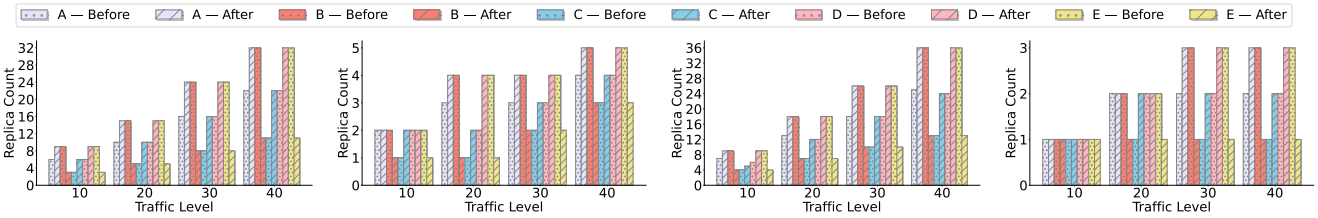
Fig. 5: Response time under varying traffic and intent

Cost: Figure 7(a) shows application cost in USD/h across traffic levels 10, 20, 30, and 40, with traffic level on the x-axis and cost on the y-axis. Colors indicate intents A-E. Cost is prioritized in B and E; after these submissions, cost decreases consistently at all traffic levels, showing that the system adapts provisioning and placement to meet cost objectives as workload intensity grows.

Power Consumption: Figure 7(b) shows application power consumption across traffic levels, with traffic level on the x-axis and power (Watts) on the y-axis. Power is prioritized in intent C, but C also emphasizes reliability, shifting replicas toward cloud nodes and slightly increasing total power. This reflects an intended trade-off between power and reliability rather than a failure to satisfy the intent.

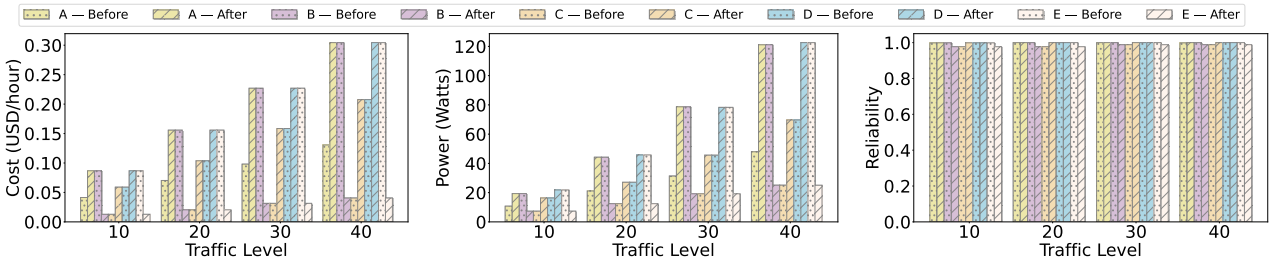
Reliability: Figure 7(c) shows application reliability across traffic levels, with traffic level on the x-axis and reliability in [0,1] on the y-axis. Reliability is prioritized in intents A, C, and D; after these submissions, reliability increases consistently at all traffic levels. This gain is driven by placing a larger share of replicas on cloud nodes with lower failure probabilities.

Accuracy: To evaluate *CognitiveContinuum*'s intent-interpretation accuracy, we used 100 GPT-4o-generated intents for the image-processing application. Each intent was manually labeled with ground-truth priorities for response time, cost, power, and reliability, then re-submitted to GPT-4o for comparison. For example, the intent *"The system must immediately recognize faces at the door and alert users. Power isn't a concern since it's always plugged in. The user expects it to work most of the time, but some false alarms or crashes are okay. It should be affordable."* maps to **response time** →***"high"***, **cost** →***"high"***, **power** →***"low"***, and **reliability** →***"medium"***. Table III compares expected and GPT-4o-generated metrics for 100 intents, counting a



(a) Microservice 1 (b) Microservice 2 (c) Microservice 3 (d) Microservice 4

Fig. 6: Replica counts per microservice across traffic levels, with each color representing a specific intent (A–E).



(a) Application cost (b) Application power consumption (c) Application reliability

Fig. 7: Performance metrics pre- (dotted) and post-intent (hashed) under varying traffic, with colors denoting different intents.

mismatch when any metric differs. Only 5 intents mismatched (2 response time, 3 power), yielding 95% accuracy and confirming *CognitiveContinuum*'s reliable mapping of natural-language intents to performance metrics.

metric	Response Time	Cost	Power	Reliability	Overall
Accuracy	98%	100%	97%	100%	95%

TABLE III: Expected vs. GPT-4o accuracy across 100 intents.

IV. Related Work

Workload orchestration across the edge–cloud continuum is rapidly advancing, with research broadly classified into four key themes discussed below.

Fuzzy-Logic Orchestration: Fuzzy reasoning is widely used to handle uncertainty in edge task placement. RL-enhanced fuzzy trees support adaptive orchestration under IoT heterogeneity [18], and TSFTO reports 46.66% lower execution time and 10% lower energy use [19]. Intuitionistic fuzzy sets and MEC fuzzy offloading further improve reliability, QoS, and resource utilization [20], [21].

Intent-Driven Orchestration: IDO [22] lets users specify high-level goals instead of low-level parameters. With Kubernetes, it enables SLO-based automation [22], [23], but intent translation remains challenging in heterogeneous, dynamic continuum environments [24]. Oakestra [25], SLOC [26], and IntentContinuum [3] improve scaling/placement and reduce QoS, cost, and energy issues.

AI and Learning-Based Orchestration: AI-driven orchestration is increasingly important for LLM workloads. Kubernetes extensions now use AI-aware scheduling and autoscaling to map goals such as latency and cost to resource decisions [27]. Edge-LLM [28] and distributed frameworks (DeepSpeed [29], PETALS [30]) improve efficiency and collaboration, while broader ML/DL/RL work supports forecasting, anomaly detection, and adaptive orchestration.

Decentralized Orchestration Frameworks: Decentralized orchestration distributes control to improve scalability and reduce bottlenecks in edge/IoT systems [31]. Swarm intelligence enables robust, energy-efficient coordination, and semantic models improve interoperability [32], [33], [34], [35]. However, these approaches are still studied separately, and tighter integration remains open [31], [36], [37].

V. Conclusions and Future Work

In this paper, we introduced *CognitiveContinuum*, an intent-aware orchestration framework that combines large language models and fuzzy reasoning to bridge the gap between vague user goals and concrete deployment actions across the compute continuum. By shifting orchestration from low-level directives to human-centric intent, the framework enables adaptive, context-aware, and accurate resource management while integrating seamlessly with widely used platforms such as Kubernetes and SDN controllers. The performance evaluation demonstrates that *CognitiveContinuum* is not only feasible but also highly effective. It reliably interprets user input, dynamically balances trade-offs, and consistently fulfills user-defined objectives under diverse and dynamic workloads. These results confirm its potential as a practical and intelligent tool for real-world edge–cloud orchestration. In future work, we aim to expand the supported objectives, improve resource efficiency by combining fuzzy reasoning with optimization techniques, and extend the framework from a single-user model to multi-user environments with multiple, potentially conflicting objectives through intent resolution and prioritization.

ACKNOWLEDGEMENTS

Akbari is supported by a Faculty of IT PhD scholarship. Grundy is supported by ARC Laureate Fellowship FL190100035. Toosi is supported by ARC through funded projects ARC DP230100081 and ARC LP210200213.

REFERENCES

[1] T. He, A. N. Toosi, N. Akbari, M. T. Islam, and M. A. Cheema, "An intent-based framework for vehicular edge computing," in *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2023, pp. 121–130.

[2] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[3] N. Akbari, J. Grundy, A. Cheema, and A. N. Toosi, "Intentcontinuum: Using llms to support intent-based computing across the compute continuum," in *2025 IEEE International Conference on Web Services (ICWS)*, 2025, pp. 573–583.

[4] A. Al-Dulaimy, M. Jansen, B. Johansson, A. Trivedi, A. Iosup, M. Ashjaei, A. Galletta, D. Kimovski, R. Prodan, K. Tserpes *et al.*, "The computing continuum: From iot to the cloud," *Internet of Things*, vol. 27, p. 101272, 2024.

[5] J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," *Computer Communications*, vol. 209, pp. 120–150, 2023.

[6] W. Rao and H. Li, "Energy-aware scheduling algorithm for microservices in kubernetes clouds," *Journal of Grid Computing*, vol. 23, no. 1, p. 2, 2025.

[7] S. N. Agos Jawaddi, A. Ismail, M. S. Sulaiman, and V. Cardellini, "Analyzing energy-efficient and kubernetes-based autoscaling of microservices using probabilistic model checking," *Journal of Grid Computing*, vol. 23, no. 1, p. 3, 2025.

[8] M. Bendechache, S. Svorobej, P. Takako Endo, and T. Lynn, "Simulating resource management across the cloud-to-thing continuum: A survey and future directions," *Future Internet*, vol. 12, no. 6, p. 95, 2020.

[9] A. Singh, G. S. Aujla, and R. S. Bali, "Intent-based network for data dissemination in software-defined vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5310–5318, 2020.

[10] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 625–655, 2022.

[11] P. Cintula, C. G. Fermüller, and C. Noguera, "Fuzzy logic," 2016.

[12] A. N. Toosi and R. Buyya, "A fuzzy logic-based controller for cost and energy efficient load balancing in geo-distributed data centers," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 186–194.

[13] 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)*. IEEE, 2024, pp. 468–474.

[14] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[15] S. Alamouti, "Quantifying energy and cost benefits of hybrid edge cloud: Analysis of traditional and agentic workloads," *arXiv preprint arXiv:2501.14823*, 2025.

[16] E. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog, and edge computing infrastructures," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 2, pp. 277–288, 2019.

[17] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao *et al.*, "Predicting node failure in cloud service systems," in *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 480–490.

[18] C. Mechalikh, H. Taktak, and F. Moussa, "A fuzzy decision tree based tasks orchestration algorithm for edge computing environments," in *International Conference on Advanced Information Networking and Applications*. Springer, 2020, pp. 193–203.

[19] L. Kheroua, Z. Doukha, and S. Moussaoui, "Tsfto: A two-stage fuzzy-based tasks orchestration algorithm for edge and fog computing environments," in *International Conference on Computing and Communication Networks*. Springer, 2023, pp. 53–62.

[20] C. Huang, B. Fan, and C. Jiang, "A task orchestration strategy in a cloud-edge environment based on intuitionistic fuzzy sets," *Mathematics*, vol. 12, no. 1, p. 122, 2023.

[21] V. Nguyen, T. T. Khanh, T. D. Nguyen, C. S. Hong, and E.-N. Huh, "Flexible computation offloading in a fuzzy-based mobile edge orchestrator for iot applications," *Journal of Cloud Computing*, vol. 9, no. 1, p. 66, 2020.

[22] T. Metsch, M. Viktorsson, A. Hoban, M. Vitali, R. Iyer, and E. Elmroth, "Intent-driven orchestration: Enforcing service level objectives for cloud native deployments," *SN Computer Science*, vol. 4, no. 3, p. 268, 2023.

[23] Y. Sharma, D. Bhamare, N. Sastry, B. Javadi, and R. Buyya, "Sla management in intent-driven service management systems: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–38, 2023.

[24] J. Spillner, J. F. Borin, and L. F. Bittencourt, "Intent-based placement of microservices in computing continuums," in *Future Intent-Based Networking: On the QoS Robust and Energy Efficient Heterogeneous Software Defined Networks*. Springer, 2021, pp. 38–50.

[25] G. Bartolomeo, M. Yosofie, S. Bäurle, O. Haluszczynski, N. Mohan, and J. Ott, "Oakestra: A lightweight hierarchical orchestration framework for edge computing," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 215–231.

[26] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, and Y. Xiong, "Sloc: Service level objectives for next generation cloud computing," *IEEE Internet Computing*, vol. 24, no. 3, pp. 39–50, 2020.

[27] Z. Ye and R. Ying, "An ai-aware orchestration framework for cloud-based llm workloads," in *2024 IEEE 10th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2024, pp. 22–24.

[28] F. Cai, D. Yuan, Z. Yang, and L. Cui, "Edge-llm: A collaborative framework for large language model serving in edge computing," in *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 2024, pp. 799–809.

[29] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3505–3506.

[30] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals: Collaborative inference and fine-tuning of large models," *arXiv preprint arXiv:2209.01188*, 2022.

[31] B. Anuraj, D. Calvaresi, J.-M. Aerts, and J.-P. Calbimonte, "Dynamic swarm orchestration and semantics in iot edge devices: A systematic literature review," *Ieee access*, 2024.

[32] Y. Miao, K. Hwang, D. Wu, Y. Hao, and M. Chen, "Drone swarm path planning for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 5, pp. 6836–6848, 2022.

[33] H. Singh, A. Bhasin, and P. R. Kaveri, "Qras: efficient resource allocation for task scheduling in cloud computing," *SN Applied Sciences*, vol. 3, no. 4, p. 474, 2021.

[34] A. S. Thuluva, D. Anicic, S. Rudolph, and M. Adikari, "Semantic node-red for rapid development of interoperable industrial iot applications," *Semantic Web*, vol. 11, no. 6, pp. 949–975, 2020.

[35] P. C. Calcina-Ccori, L. C. C. De Biase, G. Fedrecheski, F. S. C. da Silva, and M. K. Zuffo, "Enabling semantic discovery in the swarm," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 1, pp. 57–63, 2018.

[36] M. Razian, M. Fathian, R. Bahsoon, A. N. Toosi, and R. Buyya, "Service composition in dynamic environments: A systematic review and future directions," *Journal of Systems and Software*, vol. 188, p. 111290, 2022.

[37] R. C. Sofia, D. Dykeman, P. Urbanetz, A. Galal, and D. A. Dave, "Dynamic, context-aware cross-layer orchestration of containerized applications," *IEEE Access*, vol. 11, pp. 93 129–93 150, 2023.