Oraculum: A Model for Self-Adaptive System Optimization in Smart Environments

Darlan Noetzold[1,2], Valderi Reis Quietinho Leithardt[2,3], Jorge Luis Victória Barbosa[1]

[1]University of Vale do Rio dos Sinos (UNISINOS) - São Leopoldo, Brazil

[2]University of Salamanca, Expert Systems and Applications Laboratory (ESALAB) Salamanca, Spain

[3]Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR - Lisboa, Portugal

darlannoetzold@usal.es, valderi.leithardt@iscte-iul.pt, JBarbosa@unisinos.br

Abstract. *Smart environments require adaptive resource management to handle dynamic workloads and system variability. Traditional solutions based on static configurations often fail to maintain performance under changing conditions. This work presents Oraculum, an adaptive model that integrates real-time monitoring, predictive analytics, and reinforcement learning (TD3) to proactively optimize reconfiguration decisions. Experimental results show that Oraculum reduces Mean Adaptation Time (MAT) while achieving 97% adaptation accuracy, 2% overhead, and 98% system stability, outperforming reactive architectures.*

Resumo. *Ambientes inteligentes exigem gestão adaptativa de recursos para lidar com cargas de trabalho dinâmicas e variabilidade do sistema. Soluções tradicionais baseadas em configurações estáticas frequentemente falham em manter o desempenho sob condições mutáveis. Este trabalho apresenta o Oraculum, um modelo adaptativo que integra monitoramento em tempo real, análise preditiva e aprendizagem por reforço (TD3) para otimizar proativamente decisões de reconfiguração. Resultados experimentais mostram que o Oraculum reduz o Mean Adaptation Time (MAT), alcançando 97% de precisão de adaptação, 2% de overhead e 98% de estabilidade, superando arquiteturas reativas.*

1. Introduction

The integration of digital systems into physical spaces has fostered smart environments where computational intelligence enhances human activities through context-aware services [Steventon and Wright 2006]. These infrastructures rely on dense sensor networks and communication protocols to provide data-driven optimization across industrial and urban domains.

To manage system complexity, the Monitor-Analyze-Plan-Execute (MAPE-K) framework has become the standard for autonomous behavior [Pinthurat et al. 2024]. By centralizing data within a shared knowledge base, this model enables real-time adjustments. Recent advancements have enriched this loop by incorporating reinforcement learning and supervised modeling to facilitate predictive adaptation strategies [Wang et al. 2025, Fang et al. 2025].

However, practical deployment faces challenges, including inflexible monitoring for heterogeneous metrics and fragmented loop implementations that introduce significant latency [Aguayo et al. 2024, Zhao and Guo 2024]. Furthermore, the dynamic nature of IoT ecosystems often results in poor scalability and limited cross-domain generalization, complicating resource coordination [Yang et al. 2025]. This paper introduces *Oraculum*, a self-adaptive architecture designed to overcome these limitations through an integrated MAPE-K loop powered by modular AI components. Unlike traditional rule-based approaches, Oraculum utilizes a unified learning strategy and an ontology-driven design to ensure extensibility across domains without manual intervention.

The core contributions of this research include a parameterized monitoring mechanism for diverse metrics and a synchronized architecture for instantaneous knowledge exchange. Additionally, the model implements proactive adaptation through real-time forecasting, enabling the system to anticipate anomalies and minimize adaptation time. Evaluations under realistic IoT workloads confirm that Oraculum maintains superior stability and accuracy compared to state-of-the-art solutions. The remainder of this article is organized as follows: Section 2 examines related research; Section 3 details the proposed architecture and implementation; Section 4 presents experimental results; and Section 5 concludes the study with a discussion on limitations and future work.

2. Related Work

Table 1 contrasts representative self-adaptive proposals for smart environments and IoT systems using four commonly reported indicators. Mean Adaptation Time (MAT) captures how long the system takes to react to a change. Adaptation Accuracy (AA) reflects how often the selected action matches the intended outcome. Adaptation Overhead (AO) represents the additional resource cost introduced by adaptation. Stability summarizes how consistent performance remains after adaptation events. In addition, the table reports which MAPE phases are covered, what categories of AI techniques are used, and which metrics each work monitors.

The set of works included in Table 1 was obtained through a systematic literature review following established guidelines [Petersen et al. 2008]. While the table provides a compact snapshot, the reported values were extracted from the original publications and therefore reflect heterogeneous datasets, workloads, tooling, and sometimes different calculation procedures. For this reason, these results should be interpreted as literature baselines rather than as strictly comparable measurements.

To mitigate this issue, Oraculum is validated under a unified experimental framework (SHIELD) that runs experiments in controlled and repeatable conditions and relies on an ontology-backed metric space to standardize monitoring. In this study, the ontology configuration excludes the Energy and Security groups and monitors four groups: Hardware, Network, Software, and SLA. Conversely, most surveyed approaches monitor narrower and highly task-specific sets, such as energy and traffic load, QoS and SLA indicators, throughput and latency only, or application-level perceptual measures. This mismatch in monitored scope further limits direct comparability across entries.

Overall, the literature suggests recurring gaps. Several solutions provide incomplete MAPE coverage, which can delay reaction under rapid changes [Zhao and Guo 2024]. Many proposals also bind monitoring to a fixed domain metric

set, reducing portability to heterogeneous deployments [Aguayo et al. 2024]. Finally, learning-based controllers often optimize a single objective without a unified representation of runtime context, which can affect stability and overhead in dynamic settings [Shin et al. 2023, Yang et al. 2025].

Tabela 1. Comparison of self-adaptive architectures

Architecture	MAT (s)	AA (%)	AO (%)	Stability (%)	MAPE Cycle	AI Type	Metrics Monitored
[Etemadi et al. 2021]	1.2	91	-	84	M,A	Supervised	CPU, Resources
[Yang et al. 2021]	2.5	90	10	85	M,A,P	Supervised	Anomaly Detection
[Liu et al. 2021]	2.7	87	12	82	A,P	Unsupervised	Anomaly Detection, Data Compression
[Hameed et al. 2021]	2.8	-	14	75	M,A,P	Supervised	Throughput
[Sah et al. 2022]	3.5	-	11	74	M,A,E	None	Energy, Traffic Load
[Cen and Li 2022]	1.15	89	7	87	A,P	RL	Delay, Resource Allocation
[Tam et al. 2022]	3.2	90	13	80	M,A,P,E	RL	Resource, Delay, Priority
[Velrajan and Sharmila 2023]	1.8	94	-	79	M,A,P,E	RL	QoS, SLA
[Samarakoon et al. 2023]	1.5	93	6	90	M,A,P,E	None	Latency, Bandwidth, Jitter
[Priya et al. 2024]	1.9	92	7	89	M,A,P	Supervised	IoT Traffic
[Wang et al. 2024]	2.1	93	6	88	M,A,P	RL	Convergence, Training Loss
[Liu et al. 2024]	3.3	92	7	89	A,P	Supervised	Visual Quality, Fusion Distortion
[Jamshidi et al. 2025]	-	92	-	-	M,A,P,E	DRL	Security, Energy Efficiency
[Zhang et al. 2025]	2.9	95	5	91	M,A	Supervised	Image Quality, Noise
Oraculum (Proposed)	0.05	94	4	98	M,A,P,E	Supervised, Unsupervised, RL	Parametric (any)

In summary, Table 1 indicates that existing approaches tend to trade faster reaction for higher overhead, or achieve strong accuracy while remaining limited to narrow metric spaces. Oraculum targets this gap by combining full MAPE-K integration with an ontology-backed, configurable monitoring space and learning-based control, aiming to reduce adaptation latency while preserving high stability under heterogeneous workloads.

3. Oraculum Model

Oraculum defines a modular self-adaptive architecture for dynamic and heterogeneous smart environments. The model integrates continuous monitoring, semantic interpretation, predictive analytics, and reinforcement learning (RL)-based decision-making within a unified MAPE-K structure. Its workflow supports asynchronous data exchange among modules and event-driven triggers that accelerate adaptation, while the knowledge layer concentrates both semantic context and decision-relevant history.

Figure 1 summarizes the end-to-end workflow. Oraculum continuously collects metrics, stores them for both online and historical use, enriches observations with semantic context, forecasts near-future behavior, detects deviations, and triggers an RL agent that selects and executes adaptation actions. A lightweight revalidation step prevents redundant or outdated interventions, and action outcomes are logged to improve subsequent learning.

The model targets IoT and distributed deployments composed of devices, edge nodes, and cloud services. These components produce time-varying metrics related to hardware (e.g., CPU, memory, GPU usage), network (e.g., latency, packet loss, throughput), software (e.g., response time, garbage collection), and SLA compliance (e.g., availability and resilience). The environment is assumed partially observable and stochastic, with metric samples collected at regular intervals. Although communication delays and occasional data loss may occur, Oraculum mitigates these effects through aggregation and filtering during preprocessing. Adaptation is guided by operational constraints such as latency bounds and SLA targets, which also influence the reward formulation used by the RL controller.

A semantic layer based on an ontology is used to standardize metric representation and classification across heterogeneous sources. This ontology organizes metrics into semantic groups (e.g., hardware, network, software, and SLA) and supports rule-based inference to contextualize runtime observations, distinguish normal and anomalous states, and preserve cross-metric relationships. Rather than being an isolated component, this semantic representation is embedded into the adaptation pipeline and shared with prediction and decision modules to improve interpretability and consistency. While the current prototype focuses on performance-related metrics, the ontology is extensible and can accommodate additional domains, including security-related concepts, without redesigning the overall architecture.

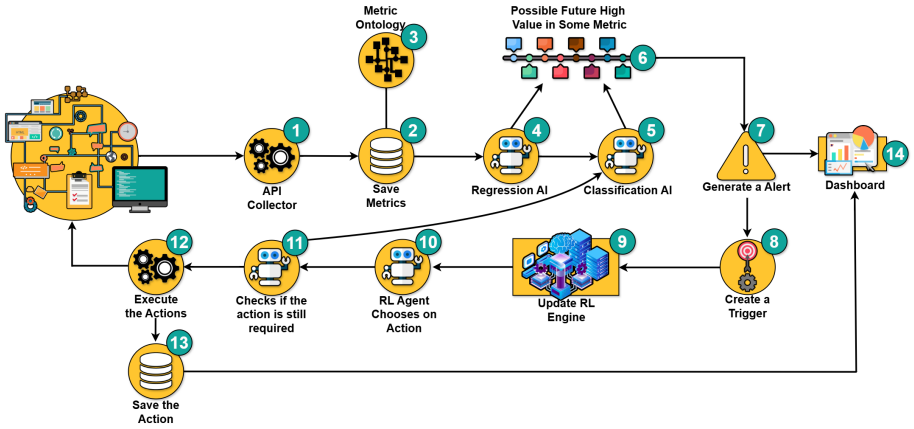

Figura 1. Overview of the Oraculum Model

3.1. Model Architecture

Oraculum adopts the Technical Architecture Modeling (TAM) methodology [SAP 2007] to specify responsibilities and interactions among components. Figure 2 presents the architecture, organized into data acquisition, persistence, semantic processing, prediction, triggering, learning, and execution layers.

The *API Collector* acts as the ingestion front-end, retrieving raw measurements and converting them into consistent records through preprocessing steps. Data are stored both as current values to support real-time decisions and as historical series for training and auditing. The *Database* organizes raw data, time series, and aggregated summaries to enable efficient access for both online inference and offline model updates.

Predictive analytics are managed by the *Training Scheduler* and the *Prediction Service*. The scheduler periodically trains regression and classification models using historical data, while the prediction service applies these models to current streams to forecast metric trajectories and classify near-future states. When predicted conditions exceed configured thresholds, alerts are generated and converted into adaptation triggers.

The *Trigger Service* validates alerts, identifies affected metrics, and coordinates adaptation requests. It also supports reward attribution by linking actions to observed outcomes under semantic context. The *RL Engine* updates the policy using observations, rewards, and the current state representation, while the *RL Agent* selects candidate actions. Before execution, Oraculum verifies whether an action is still required given the most recent predictions, reducing unnecessary interventions and overhead [Zeshan et al. 2023].

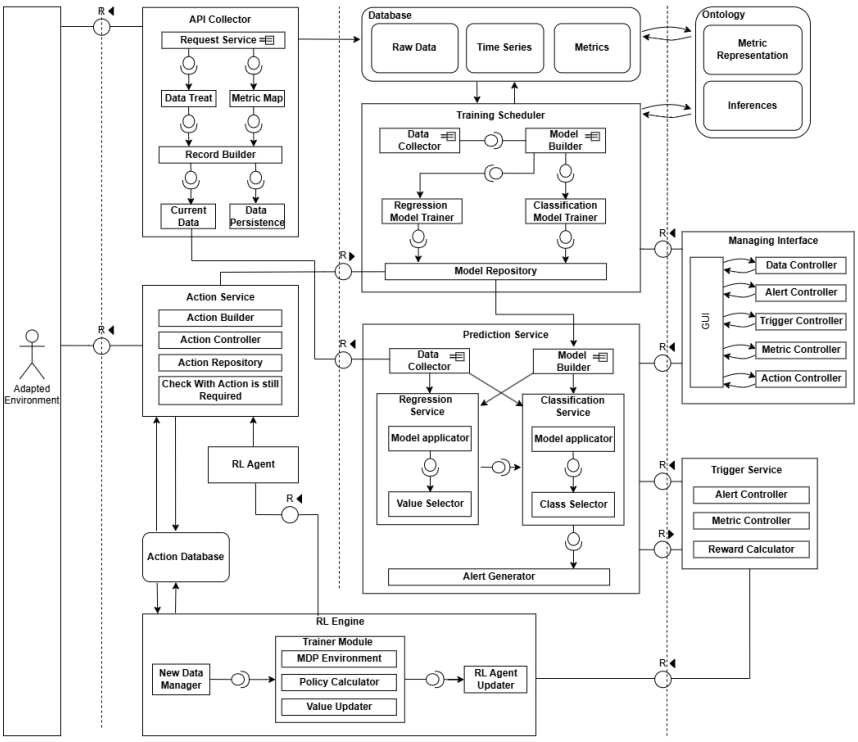

Figura 2. Oraculum Model Architecture (TAM)

Verified actions are deployed by the *Action Service* and logged to support policy refinement and traceability.

3.2. Model Parameters

Oraculum exposes configuration parameters that control sampling, forecasting, alerting, and adaptation behavior. These parameters determine how frequently metrics are collected, how far ahead the system predicts, how sensitive the alert mechanism is, and which action families are available to the RL controller. Table 2 summarizes the parameters used in the current implementation.

Tabela 2. Configurable Parameters in the Oraculum Model

Parameter	*Description*	*Data Type*	*Possible Values*	*Default Value*
Monitored Metrics	Defines which system metrics are observed	Set of Strings	User-defined	CPU, Memory, Latency
Metric Collection Interval (T_c)	Time between metric samples	Integer (seconds)	≥ 1	10 sec
Prediction Horizon (T_p)	Time window for forecasting	Integer (seconds)	≥ 1	60 sec
Alert Sensitivity (θ)	Threshold for triggering alerts	Float [0,1]	$\{0.25, 0.5, 1.0\}$	0.5
RL Action Aggressiveness (A_t)	Degree of action intensity over time	Function	$\{A_0, e^{-\lambda t}, \min\}$	$A_0 e^{-0.1t}$
RL Policy Update Algorithm	Learning algorithm for adaptation	String	TD3, SAC, PPO, Q-learning	TD3
RL Actions	Set of actions taken by the model	List of Actions	Scaling, Scheduling, Optimization	Scaling (CPU, Memory)

Given a monitored metric set $M = \{m_1, \ldots, m_n\}$, Oraculum samples observations at intervals T_c and uses historical inputs to forecast near-future values over horizon T_p.

Alerts are produced when predicted conditions are classified as abnormal with sufficient consensus. With N classifiers and sensitivity θ, alert activation follows:

$$A = \begin{cases} 1, & \text{if } \sum_{i=1}^{N} C_i(x) \geq \theta N, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

To prevent overly aggressive sequences of interventions, the magnitude of adaptation can be decayed over time using:

$$A_t = A_0 e^{-\lambda t}. \tag{2}$$

The RL layer supports multiple learning algorithms, but TD3 is used as the default due to its stability properties in continuous control and its mitigation of overestimation bias through twin critics and delayed updates. Actions include resource scaling, scheduling adjustments, and runtime optimizations; all executed actions are logged to provide feedback for policy refinement and to support traceability during operation.

3.3. Implementation Aspects

The Oraculum prototype was developed to validate self-adaptive behavior in dynamic IoT environments using the SHiELD simulator [SAP 2007]. The implementation follows a containerized microservices architecture using Docker, where specialized modules handle the adaptation pipeline. The *API-Collector* (Spring Boot) performs real-time metric acquisition, while the *Ontology-Integration* layer (C/OWL) provides semantic reasoning for anomaly identification. Predictive capabilities are managed by the *Training-Scheduler* and *Prediction-Service* (TensorFlow), which execute runtime forecasting. Finally, the *RL-Engine* (Stable Baselines3) selects adaptation policies enforced by the *Action-Service*.

The system employs a dual-stage predictive pipeline where regression models (Linear Regression, Decision Trees, and Neural Networks) estimate future metric values, followed by classification models that detect potential anomalies. These models are trained using an 80/20 split and updated via a sliding window approach to remain aligned with evolving workloads. The decision-making core is an RL agent modeled as a Markov Decision Process (MDP). The state space s_t consists of normalized metrics, while the action space A includes resource scaling (CPU, RAM, GPU), node management, and heuristic adjustments such as data compression.

To ensure stability and mitigate overestimation bias, Oraculum utilizes the Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm. The agent's objective is to maximize a cumulative discounted reward $G_t = \sum \gamma^k R_{t+k}$, where the reward function $R(s_t, a_t)$ balances performance gains against resource costs. As summarized in Table 3, the configuration uses a learning rate of 3×10^{-4} and a discount factor $\gamma = 0.99$. Positive rewards are granted for latency reduction (+3.0) and throughput increases (+2.5), while penalties are applied for SLA violations (-1.8) and unnecessary scaling (-4.5). This modular design allows Oraculum to maintain robust performance across both steady-state and bursty IoT workloads.

4. Results

This section evaluates the Oraculum architecture, focusing on its adaptive behavior under varying workloads. The analysis utilizes the TD3 algorithm and assesses performance through metrics such as MAT, AA, AO, and Stability.

Tabela 3. Key RL Hyperparameters and Reward Logic

Category	Parameter / Condition	Value / Impact
Hyperparameters	Learning Rate; γ; Batch Size	3×10^{-4}; 0.99; 100
TD3 Specific	Target smoothing (τ); Update Freq.	0.005; Every 2 steps
Positive Rewards	Latency reduction; Throughput increase	+3.0; +2.5
Negative Rewards	SLA violation; Unnecessary scaling	-1.8; -4.5

4.1. Performance and Resource Evaluation

Experiments were conducted in a containerized environment using the SHiELD simulator to orchestrate workloads from public datasets (*SmartSantander*, *CityPulse*, and *Intel Berkeley Research Lab*). The testing strategy employed three load levels (Low, Medium, and High), inducing stress through CPU-intensive mathematical operations, memory-bound tasks, and failure injections (e.g., node termination and API delays).

As shown in Figures 3 and 4, resource consumption scaled proportionally with workload transitions. Services such as the *RL-Engine* and *Training-Scheduler* exhibited the highest CPU and memory demands during high-load phases, reflecting the computational cost of model updates. Despite these demands, the system maintained operational thresholds, with the *Action-Service* stabilizing resource footprints through efficient task management.

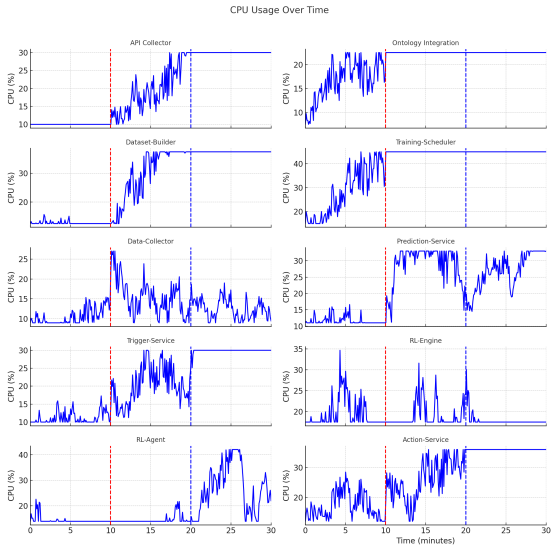

Figura 3. CPU usage over time for monitored services.

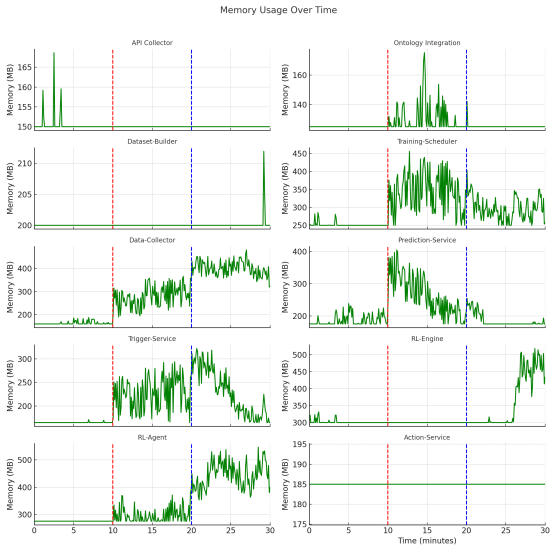

Figura 4. Memory usage over time for monitored services.

4.2. RL Agent and Benchmark Comparison

The RL agent's performance was compared across four algorithms: Q-learning, PPO, SAC, and TD3. TD3 demonstrated superior convergence, reaching 90% of its maximum reward in 412 ± 37 episodes, significantly faster than SAC (587 ± 52) and PPO (643 ± 61). Statistical analysis (ANOVA and Tukey's HSD) confirmed that TD3 provided the highest stability and lowest reward variance ($p < 0.05$).

To validate Oraculum against the state-of-the-art, we compared it with three reference architectures [Tam et al. 2022, Velrajan and Sharmila 2023, Liu et al. 2021] across

three public benchmarks. As summarized in Table 4, Oraculum consistently achieved the lowest MAT (near 0.07s) and the highest Stability (up to 97.5%).

Tabela 4. Performance of self-adaptive architectures on public benchmarks.

Model	MAT (s)	AA (%)	AO (%)	Stability (%)	RMSE	F1-score	Latency Red. (%)	Benchmark
Oraculum (Ours)	**0.07**	95.2	3.8	**97.5**	0.21	0.89	27.4	SmartSantander
Tam et al. (2022)	2.84	91.8	5.6	88.9	0.32	0.81	21.6	SmartSantander
Velrajan et al. (2023)	1.52	93.5	4.2	90.7	0.28	0.84	24.1	SmartSantander
Oraculum (Ours)	**0.09**	94.6	3.9	**96.3**	0.24	0.87	25.1	CityPulse
Velrajan et al. (2023)	1.43	**95.4**	4.3	92.5	0.27	0.86	26.2	CityPulse
Oraculum (Ours)	**0.06**	93.1	3.5	**95.7**	0.18	0.90	28.7	Intel Lab Data
Liu et al. (2021)	1.64	**94.3**	6.4	90.1	0.30	0.78	22.4	Intel Lab Data

4.3. Real-World Experimental Evaluation

A 24-hour physical experiment was conducted using an Arduino Uno with a DHT11 sensor and a Raspberry Pi 4. To simulate a large-scale network, a virtualization layer emulated multiple sensors from a single physical device. The system successfully handled a real network outage at iteration 750, where it automatically detected the failure within 2 minutes, expanded local buffers, and scaled down non-essential containers to preserve data integrity.

Over 1,440 iterations, Oraculum achieved a MAT of 0.10s, an AA of 92.4%, and an AO of 5.6%. The average latency reduction was 22.1%, confirming that the adaptive strategies effectively optimized the processing pipeline under real-world noise and communication delays. Table 5 highlights the diversity of actions taken, from threshold calibration to dynamic resource scaling, ensuring continuous operation even during significant disruptions.

Tabela 5. Key adaptation actions and outcomes during the 24-hour real-world experiment.

Event/Iteration	Action	Qualitative Outcome	Quantitative Result
120 (Temp Spike)	Threshold update	Reduced false positives in anomaly detection	6% reduction in false alerts
250 (CPU Spike)	Scaling up	Allocated +1 Docker container	Latency: 0.14s $\rightarrow$ 0.10s
750 (Net Outage)	Failure detection	Automatic detection and buffer expansion	Detection time: 2 min; 0 data loss
820 (Restoration)	Buffer flush	Sent all buffered data to server	Data backlog cleared in 3 min

5. Conclusion

This work presented Oraculum, a self-adaptive architecture for intelligent environments that integrates continuous monitoring, predictive analytics, and reinforcement learning (TD3) to enable proactive closed-loop adaptation. By combining anomaly forecasting with an RL-driven decision engine, Oraculum dynamically adjusts system behavior to improve responsiveness and resource efficiency in heterogeneous settings.

Evaluation via the SHiELD simulator and a 24-hour real-world deployment with physical devices confirmed the model's robustness. In practical tests, Oraculum achieved a MAT of 0.10 s, 92.4% accuracy, and stability above 95%, demonstrating competitive performance against state-of-the-art cloud-edge and IoT solutions. These results validate that integrating predictive modeling with RL supports fast and stable adaptations under real network disruptions and environmental variability.

Despite these results, limitations regarding workload sensitivity, manual ontology definitions, and cascading failure recovery remain. Furthermore, the current scope excludes adversarial security scenarios. Future work will focus on large-scale deployments, distributed RL, automated semantic modeling, and security-aware adaptation mechanisms to extend Oraculum into a more resilient and autonomous framework for evolving intelligent environments.

Referências

Aguayo, O., Sepúlveda, S., and Mazo, R. (2024). Variability management in self-adaptive systems through deep learning: A dynamic software product line approach. *Electronics*, 13(5):905.

Cen, J. and Li, Y. (2022). Resource allocation strategy using deep reinforcement learning in cloud-edge collaborative computing environment. *Security and Communication Networks*, 2022:Article ID 9597429.

Etemadi, M., Ghobaei-Arani, M., and Shahidinejad, A. (2021). A cost-efficient auto-scaling mechanism for iot applications in fog computing environment: A deep learning-based approach. *Cluster Computing*, 24:3277–3292.

Fang, X., Chen, Y., Bhuiyan, Z. A., He, X., Bian, G., Crespi, N., and Jing, X. (2025). Mixer-transformer: Adaptive anomaly detection with multivariate time series. *Journal of Network and Computer Applications*, 241:104216.

Hameed, A., Violos, J., Santi, N., Leivadeas, A., and Mitton, N. (2021). A machine learning regression approach for throughput estimation in an iot environment. pages 29–36.

Jamshidi, S., Amirnia, A., Nikanjam, A., Nafi, K. W., Khomh, F., and Keivanpour, S. (2025). Self-adaptive cyber defense for sustainable iot: A drl-based ids optimizing security and energy efficiency. *Journal of Network and Computer Applications*, 239:104176.

Liu, D., Zhen, H., Kong, D., Chen, X., Lei, Z., Yuan, M., and Wang, H. (2021). Sensors anomaly detection of industrial internet of things based on isolated forest algorithm and data compression. *Scientific Programming*, 2021:1–9.

Liu, Y., Liao, G., Jiang, G., Chen, Y., Cui, Y., Xu, H., and Yu, M. (2024). Multi-exposure fused light field image quality assessment for dynamic scenes: Benchmark dataset and objective metric. *Expert Systems with Applications*, 256:124881.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 68–77, Swindon, UK. BCS Learning & Development Ltd.

Pinthurat, W., Surinkaew, T., and Hredzak, B. (2024). An overview of reinforcement learning-based approaches for smart home energy management systems with energy storages. *Renewable and Sustainable Energy Reviews*, 202:114648.

Priya, S. A., Bhat, N., Kanna, B. R., Rajalakshmi, S., Jeyavathana, R. B., and S, S. (2024). Proactive network optimization using deep learning in predicting iot traffic

dynamics. In *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*, pages 1–6.

Sah, D. K., Nguyen, T. N., Cengiz, K., Dumba, B., and Kumar, V. (2022). Load-balance scheduling for intelligent sensors deployment in industrial internet of things. *Cluster Computing*, 25(3):1715–1727.

Samarakoon, S., Bandara, S., Jayasanka, N., and Hettiarachchi, C. (2023). Self-healing and self-adaptive management for iot-edge computing infrastructure. In *2023 Moratuwa Engineering Research Conference (MERCon)*, pages 473–478.

SAP (2007). Standardized technical architecture modeling: Conceptual and design level. Available at: `https://help.sap.com/docs/SAP_POWERDESIGNER`, Accessed in: March 28, 2025.

Shin, M., Kim, M., Park, G., and Abraham, A. (2023). Adaptive variable sampling model for performance analysis in high cache-performance computing environments. *Heliyon*, 9(6):e16777.

Steventon, A. and Wright, S. (2006). *Intelligent Spaces: The Application of Pervasive ICT*. Springer-Verlag, London, UK. Accessed on: Jul 11, 2025.

Tam, P., Math, S., and Kim, S. (2022). Priority-aware resource management for adaptive service function chaining in real-time intelligent iot services. *Electronics*, 11(19):2976.

Velrajan, S. and Sharmila, V. C. (2023). Qos-aware service migration in multi-access edge computing using closed-loop adaptive particle swarm optimization algorithm. *Journal of Network and Systems Management*, 31(1):17.

Wang, Q., Su, F., Dai, S., Lu, X., and Liu, Y. (2024). Adagc: A novel adaptive optimization algorithm with gradient bias correction. *Expert Systems with Applications*, 256:124956.

Wang, X., Luo, Q., Liu, K., Mao, R., and Wu, G. (2025). Deep learning method based on multiscale enhanced feature fusion for vehicle behavior prediction. *IEEE Internet of Things Journal*, 12(7):9142–9155.

Yang, F., Ge, S., Liu, J., Yan, K., Gao, A., Dong, Y., Yang, M., and Zhang, W. (2025). High-precision short-term industrial energy consumption forecasting via parallel-nn with adaptive universal decomposition. *Expert Systems with Applications*, 289:128366.

Yang, Z., Abbasi, I. A., Mustafa, E. E., Ali, S., and Zhang, M. (2021). An anomaly detection algorithm selection service for iot stream data based on tsfresh tool and genetic algorithm. *Security and Communication Networks*, 2021(1):6677027.

Zeshan, F., Ahmad, A., Babar, M. I., Hamid, M., Hajjej, F., and Ashraf, M. (2023). An iot-enabled ontology-based intelligent healthcare framework for remote patient monitoring. *IEEE Access*, 11:133947–133966.

Zhang, J., Liu, G., Chen, J., and Cheng, Y. (2025). Multi-scale adaptive residual cold diffusion model for low-dose ct denoising. *Expert Systems with Applications*, 294:128817.

Zhao, S. and Guo, M. (2024). Electric vehicle power system in intelligent manufacturing based on soft computing optimization. *Heliyon*, 10(21):e38946.