eoss-conf.com

ISSUE
№95

EUROPEAN OPEN
SCIENCE SPACE

COLLECTION OF SCIENTIFIC PAPERS

4TH INTERNATIONAL
SCIENTIFIC
AND PRACTICAL
CONFERENCE

SCIENTIFIC RESEARCH: EMERGING THEORIES AND PRACTICAL BREAKTHROUGHS

JULY 6-8, 2026, EDINBURGH, SCOTLAND

EUROPEAN OPEN SCIENCE SPACE

Proceedings of the 4th International Scientific and Practical Conference

"Scientific Research: Emerging Theories and Practical Breakthroughs"

July 6-8, 2026

Edinburgh, Scotland

Collection of Scientific Papers

Scotland, 2026

UDC 01.1

Collection of Scientific Papers with the Proceedings of the 4th International Scientific and Practical Conference «Scientific Research: Emerging Theories and Practical Breakthroughs» (July 6-8, 2026, Edinburgh, Scotland). European Open Science Space. 2026.

ISBN 979-8-89704-972-1 (series)
DOI 10.70286/EOSS-06.07.2026

The conference is included in the Academic Research Index ReserchBib International catalog of scientific conferences.

The conference is registered in the database of scientific and technical events of UkrISTEI to be held on the territory of Ukraine (Certificate №1072 dated 22.12.2025).

The materials of the collection are presented in the author's edition and printed in the original language. The authors of the published materials bear full responsibility for the authenticity of the given facts, proper names, geographical names, quotations, economic and statistical data, industry terminology, and other information.

ISBN 979-8-89704-972-1

EUROPEAN OPEN SCIENCE SPACE

A HYBRID RULE-BASED / MACHINE-LEARNING AUTOSCALER FOR KUBERNETES: REDUCING RESOURCE OVER-PROVISIONING ON A REAL CLUSTER

Murzha Dmytro
PhD student
Department of Cybersecurity and Information Systems
Kharkiv National University of Economics, Ukraine

Abstract. Resource allocation for microservice applications in Kubernetes is still handled, in most deployments, by reactive mechanisms. The Horizontal Pod Autoscaler (HPA) and extensions such as KEDA change the replica count only after a monitored metric has already crossed a fixed threshold, which in practice means over-provisioning under noisy load and a sluggish response when demand genuinely spikes. This paper presents HMRO (Hybrid Microservices Resource Optimizer), an autoscaler that pairs a deterministic rule-based engine with an ensemble of machine-learning load predictors whose influence on each decision is not fixed but adjusted continuously according to how accurate the predictors have recently been. HMRO was evaluated on a real Kubernetes cluster (Minikube) against the standard HPA across memory and combined CPU+memory workloads, each with three scenarios and ten iterations. HMRO reduced the average replica count (a direct proxy for over-provisioning) by 36–42% for memory (all $p < 0.05$) and 20–28% for combined workloads (significant in two of three scenarios), while triggering a comparable number of scaling actions – that is, without losing responsiveness. A comparable reduction was observed for CPU-driven workloads in an earlier evaluation on a prior prototype version. An ablation study isolates the source of the saving: it comes primarily from the asymmetric rule engine, whereas the ML component adds proactivity rather than resource reduction. We also discuss where a single-node evaluation falls short and what production-scale validation would still need to confirm.

Keywords: cloud computing, microservices, resource optimization, autoscaling, Kubernetes, hybrid approach, machine learning, ensemble prediction.

1. Introduction.

As microservice architectures have become the default for cloud-native systems, automated resource management has stopped being optional. An application built from dozens or hundreds of independently deployed services must continuously adjust its footprint to load that rarely sits still – too little, and service quality suffers; too much, and the infrastructure bill grows for no good reason.

Kubernetes' native autoscaler, the Horizontal Pod Autoscaler [1, 2], and extensions such as KEDA [3], all work the same reactive way: nothing happens until a monitored metric has already crossed its threshold. Two problems follow. First, short-term noise routinely triggers scaling actions that were never needed – commonly called

"flapping" – adding control-plane overhead. Second, because starting a new pod is not instant (typically tens of seconds), a purely reactive controller cannot get ahead of a spike; by the time it reacts, the spike has often already landed.

That gap is what pushes research toward predictive and hybrid strategies that combine the safety of threshold-based control with the lookahead that machine-learning forecasting can offer. The problem, as Section 3 details, is that most hybrid proposals take one of two shortcuts: switching bluntly between rule-based and ML modes, or fixing the weighting between them by hand at configuration time. Neither accounts for the fact that a predictor's reliability is not constant – it drifts with the workload and over time.

This paper takes over-provisioning – measured as the average number of replicas held during a workload – as the primary object of optimization, since it maps directly to cost, and evaluates the approach on a real Kubernetes cluster rather than in simulation.

2. Aim and objectives of the study.

The aim of this study is to build and evaluate a hybrid approach to microservice resource optimization that reduces over-provisioning without making the system less responsive, targeting the three gaps in existing hybrid methods identified in Section 3. Reaching that aim required four things: (1) designing an architecture in which a rule-based engine and an ML ensemble share decision-making, with the ML side's influence tied continuously to its recent forecasting accuracy rather than fixed in advance; (2) turning that architecture into a working prototype, HMRO, compatible with the existing Kubernetes autoscaling ecosystem; (3) running a comparative evaluation against the standard HPA on a real cluster across memory and combined CPU+memory workloads, backed by proper statistics, with an earlier CPU-only evaluation on a prior prototype version reported separately for context; and (4) being explicit about where the evaluation falls short so that the path toward production-scale validation is stated rather than implied.

3. Related work.

Autoscaling work for cloud and container environments tends to fall into four broad camps: static provisioning, threshold-based reactive control, machine-learning-based prediction, and hybrid methods [4, 5, 6].

Static provisioning – allocating a fixed amount of resources sized for the worst case – is still common because it is easy to reason about. It also produces the lowest average utilization of the four and cannot adapt when load actually varies.

Threshold-based reactive control, the Kubernetes HPA [1, 2] being the obvious example, watches a single metric – usually CPU – and scales once it crosses a configured line. It is easy to set up and widely used, but sensitive to how thresholds and cooldowns are tuned, and by design it only reacts once degradation has already started.

Machine-learning-based approaches supplement or replace threshold logic with load forecasting – from classical time-series models to recurrent neural networks – to make scaling proactive. They typically improve resource utilization, but that gain rides

entirely on forecast accuracy, which varies a lot by workload type. What happens once that accuracy degrades is something surprisingly few studies report.

Hybrid methods, combining reactive and predictive logic, are generally reported to beat either approach alone. But a review of the literature, including recent surveys of microservice autoscaling [7], turns up the same gaps repeatedly: existing hybrid proposals tend to use a fixed or binary weighting between the rule-based and ML components rather than one that adjusts continuously; they usually optimize a single metric, almost always CPU, rather than CPU and memory together; and they treat scale-up and scale-down as symmetric decisions even though the error costs are not – a missed scale-up hurts service quality, while an unneeded scale-up merely costs a little more. HMRO targets these three gaps directly.

Table 1. Mapping between limitations of existing hybrid approaches and HMRO design decisions.

Limitation of existing hybrid approaches	**Corresponding HMRO design decision**
Fixed or binary ML/rule-based weighting	Continuous confidence-based modulation (Section 4.4)
Single-metric (CPU-only) optimization	Separate CPU/memory monitoring with OR/AND rule logic (Section 4.2)
Symmetric scale-up/scale-down logic	Asymmetric thresholds and cooldown periods (Section 4.5)
No defined behaviour on ML failure	Graceful degradation to the rule-based level (Section 4.1)

4. Proposed approach.

4.1. Design principles. HMRO rests on four decisions arrived at largely by working through what goes wrong in simpler hybrid systems. Safety before efficiency: any single component requesting a scale-up gets it, while scale-down requires several components to agree – the two error types do not cost the same. Graceful degradation: each level operates independently; if the ML component fails or is unavailable, control falls back to the rule-based level and nothing breaks. Earned trust: the ML component's influence tracks its recently demonstrated accuracy rather than sitting fixed at a configured value. Self-adaptation: the system is built to adjust its own behaviour rather than depend on manual tuning every few weeks.

4.2. Three-level architecture. HMRO runs as three independent levels. Level 1, the Rule-based Engine, is the foundation – the one part guaranteed to work under any conditions. It runs deterministic threshold logic with deliberately asymmetric conditions: scale-up fires if any monitored resource is overloaded (logical OR), while scale-down requires all of them to be underutilized at once (logical AND). Level 2, the ML Predictors, switches on once roughly 20–30 observations have accumulated (about ten minutes of runtime) and handles proactive load forecasting through the ensemble (Section 4.3) and the confidence-modulation mechanism (Section 4.4). Level 3, the LLM Advisor, sits above both as a slow meta-controller: roughly every 30 minutes it reviews recent scaling behaviour and suggests parameter adjustments, without ever issuing a scaling command itself. In this study Level 3 runs in log-only mode and is

not part of the quantitative evaluation; it is described here for architectural completeness and revisited in future work.

4.3. Ensemble load prediction. The ensemble combines four predictor models chosen to cover different trade-offs between simplicity, robustness on small datasets, and the ability to capture complex temporal patterns: Linear Regression, Random Forest, ARIMA, and an LSTM network. Rather than picking one "best" model upfront, HMRO averages all four. That choice follows from preliminary benchmarks in which no single model won consistently – Random Forest had the lowest error on some datasets, Linear Regression on others – and there was no reliable way to predict in advance which would win where. A trend-correction component additionally watches for sustained upward trends (a moving-window linear fit) and applies a multiplicative correction to the ensemble output, since averaging-based predictors otherwise tend to underestimate load during sustained growth.

4.4. Confidence-based trust modulation. What separates HMRO from binary-switching hybrids is the ConfidencePredictor: it tracks the ensemble's real-world accuracy (predicted vs. observed load) and turns it into a confidence value between 0 and 1. The ML ensemble's effective influence on any given decision follows directly:

$$\text{effective\_weight} = \text{ml\_weight} \times \text{confidence} \quad (1)$$

where ml_weight is a configured base weight. When confidence is high, the ML contribution sits near that maximum; when forecasting accuracy slips, its influence shrinks automatically and smoothly, with no intervention. Under sustained degradation the system drifts back to purely rule-based behaviour.

4.5. Hybrid decision algorithm. The HybridOptimizer resolves the two levels' signals into one decision through a priority-ordered sequence of rules: (1) no sufficient ML data yet → fall back entirely to rule-based logic; (2) a rule-based scale-up request always executes (safety first); (3) if rule-based and ML agree, that joint decision executes; (4) a proactive ML-suggested scale-up fires only if the effective weight is at least 0.5, and ML blocking an unnecessary scale-down uses the same threshold; (5) an ML-suggested scale-down – the riskiest call – requires a stricter 0.6; (6) anything unresolved defaults to rule-based. Two mechanisms limit oscillation: asymmetric cooldowns (30 s after scale-up, 180 s after scale-down) and a proactive extension that lets the system scale up ahead of a threshold breach when the ensemble sees an overload coming, which helps offset the pod-provisioning delay typical of Kubernetes.

4.6. Implementation. HMRO is implemented in Python and runs alongside a standard Kubernetes cluster, monitoring the same per-pod CPU and memory metrics the native HPA uses (via Prometheus). The rule-based engine, ensemble predictors, confidence logic, and decision algorithm are each independent, testable modules – which is what makes it possible to evaluate every level separately as well as together, something the ablation study in Section 6 relies on. Whichever internal level produced a decision, the prototype exposes the same interface (scale-up / scale-down / no-op per cycle).

5. Experimental setup.

Evaluation was run on a real single-node Kubernetes cluster (Minikube, 2 vCPU, ~16 GB RAM), not in simulation. A test microservice was deployed and driven by a synthetic load generator; CPU and memory were collected through Prometheus over NodePort. HMRO was compared, scenario by scenario, against a Kubernetes HPA configured with default-style thresholds (scale-up at 80% CPU, scale-down at 20%), applied identically so the comparison is fair.

Two resource dimensions were evaluated on the same prototype version, each with three workload scenarios: memory (memory_stable, memory_spiky, memory_ramp; HPA on CPU+memory), and combined CPU+memory (combined_stable, combined_spiky, combined_ramp). An earlier evaluation of an equivalent CPU-only configuration, on a prior prototype version, is reported separately in Section 6.3 for context. Each scenario ran ten independent iterations to smooth out stochastic variation in ML training. The primary metric was the average replica count held during the run – a direct proxy for over-provisioning and hence cost. A secondary metric, the number of scaling events, was tracked to verify that any reduction in replicas did not come at the price of responsiveness. Results are reported as means with 95% confidence intervals; because HMRO and HPA are executed as independent runs, statistical significance is assessed with Welch's t-test (unequal variances), with Cohen's d as the effect size and $\alpha = 0.05$.

A note on the predictor benchmark that motivates the ensemble: tested individually across the datasets, no model dominated – Random Forest was best on some traces, Linear Regression on others – which is the empirical justification for averaging rather than committing to one model.

6. Results and discussion.

6.1. Memory workloads.

Table 2. Average replica count, memory workloads (10 iterations per scenario).

Scenario	HMRO	HPA	Reduction / p-value
memory_stable	1.00	1.67	−40%, p=0.009
memory_spiky	1.32	2.06	−36%, p=0.045
memory_ramp	1.04	1.80	−42%, p=0.002

All three memory scenarios are statistically significant. This is a direct benefit of HMRO's dual-resource monitoring: HPA scaling on a single metric tends to hold extra replicas that a joint CPU+memory view shows to be unnecessary. Scaling events over the same runs were comparable between the two controllers, so the lower footprint does not come at the cost of responsiveness.

6.2. Combined CPU+Memory workloads.

Table 3. Average replica count, combined workloads (10 iterations per scenario).

Scenario	HMRO	HPA	Reduction / p-value
combined_stable	1.42	1.98	−28%, p=0.227
combined_spiky	1.39	1.84	−24%, p=0.035
combined_ramp	1.39	1.73	−20%, p=0.004

The reduction persists under combined load, though smaller. combined_stable did not reach significance, driven by high variance in the HPA baseline (bimodal replica behaviour, sd ≈ 1.2); the effect is nonetheless in the same direction and consistent with the other two scenarios.

6.3. Earlier CPU-only evaluation. For context, an earlier evaluation of an equivalent CPU-only configuration – carried out on a prior prototype version – showed HMRO holding roughly half as many replicas as HPA across stable, spiky, and ramp scenarios (reductions of 45–47%), with a comparable number of scaling events. These results are consistent in direction and magnitude with the memory and combined results above; they are reported separately because they were produced on a different code version and are not part of the version-controlled evaluation of Sections 6.1–6.2.

6.4. Ablation analysis. Because each architectural component is an independent module, each can be disabled in isolation. Table 4 reports the effect of removing individual components on the average replica count (combined scenarios), relative to the full system, with the baseline and all ablated configurations run on the same code version.

Table 4. Ablation: effect of removing a component on average replicas (combined workloads).

Disabled component	**Effect on avg replicas**	**Interpretation**
Memory-aware scaling	+7.6%	Most valuable single component
Confidence modulation	+3.2%	Minor
ML ensemble (whole)	−2.9%	Adds proactive replicas, not savings
Adaptive weighting	0.0%	Neutral on average

The key finding is that most of the resource saving comes from the asymmetric rule engine, not from the ML component. Removing ML actually lowers the replica count (by ~3%), because the predictors proactively scale up ahead of anticipated spikes – occasionally holding replicas that a purely reactive system would not. In other words, ML in HMRO is a quality-of-service layer (fewer missed spikes, faster reaction) rather than a resource-economy layer; the economy is produced by the rule design. This reframing is more honest than the common narrative that attributes hybrid gains chiefly to the ML model, and it is only visible because the architecture supports component-level ablation.

6.5. Practical implications. Two points matter beyond the raw numbers. First, under the least predictable workloads HMRO converges to conservative rule-based behaviour rather than failing unpredictably – worst-case behaviour usually matters as much to an operator as average-case gains. Second, confidence modulation needs no per-scenario configuration, so a single deployment keeps adapting as a service's workload pattern shifts, without anyone having to reclassify the workload or retune thresholds by hand.

6.6. Limitations and threats to validity. The evaluation runs on a single-node Minikube cluster, so cross-node scheduling, node-level contention, and production-

scale image-pull and pod-startup behaviour are not exercised. Each resource dimension is covered by three base scenario types (stable, spiky, ramp); seasonal and step-function patterns are not included. Scaling is limited to ±1 replica per decision, which slows the response to very large spikes. The ML component has a cold-start period (~20 samples) before it activates. A small number of iterations were excluded by data-validation checks, so realized sample sizes are occasionally 8–9 rather than exactly ten per scenario; this does not change the direction or significance of the reported effects. The memory-workload comparison (Section 6.1) used HMRO and HPA runs recorded on adjacent, same-day commits rather than a single shared commit, unlike the combined-workload comparison (Section 6.2), which used one commit for both sides; the latter is therefore the cleanest single-commit comparison available. Latency was recorded during runs but the induced load was light, so a rigorous service-level (P95/P99, SLA-violation) analysis under production-grade load is left for future work – it is the natural next step for substantiating the quality-of-service role attributed to the ML layer in Section 6.4.

7. Conclusions and future work.

This paper described HMRO – a rule-based engine, an ML ensemble, and a confidence-modulation mechanism that ties the two together by adjusting the ML side's influence to match its demonstrated accuracy. On a real Kubernetes cluster, HMRO reduced average replica count relative to the standard HPA by 36–42% for memory (all $p < 0.05$) and 20–28% for combined workloads, while triggering a comparable number of scaling actions – that is, without sacrificing responsiveness; an earlier CPU-only evaluation on a prior version showed a consistent 45–47% reduction. An ablation study attributed the saving primarily to the asymmetric rule engine and positioned the ML layer as a source of proactivity rather than economy.

Future work has three strands: extending the evaluation to a multi-node, production-scale cluster; adding a rigorous service-level analysis (P95/P99 latency, SLA violations) under heavier load to quantify the quality-of-service contribution of the ML layer; and experimentally evaluating the LLM Advisor (Level 3), which in this study ran in log-only mode.

References.

1. Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., & Kim, S. (2020). Horizontal pod autoscaling in Kubernetes for elastic container orchestration. Sensors, 20(16), 4621. https://doi.org/10.3390/s20164621
2. Kubernetes documentation. Horizontal Pod Autoscaling. Kubernetes / Cloud Native Computing Foundation. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
3. KEDA documentation. Kubernetes Event-driven Autoscaling. Cloud Native Computing Foundation. https://keda.sh/docs/
4. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in cloud computing: State of the art and research challenges. IEEE Transactions on Services Computing, 11(2), 430–447. https://doi.org/10.1109/TSC.2017.2711009

Proceedings of the 4th International Scientific
and Practical Conference
"Scientific Research: Emerging Theories and Practical Breakthroughs"
July 6-8, 2026
Edinburgh, Scotland

Contact details of the organizing committee:
European Open Science Space
E-mail: info@eoss-conf.com
URL: https://www.eoss-conf.com/

EUROPEAN OPEN
SCIENCE SPACE