computers

MDPI

Article

A Trust-Aware Extension to a Reinforcement Learning Hyper-Heuristic Framework for Multi-Objective Scientific Workflow Scheduling

Hadeel Amjed Saeed [1,*], Sufyan T. Faraj Al-Janabi [2], Esam Taha Yassen [1] and Omar A. Aldhaibani [3]

[1] College of Computer Science and Information Technology, University of Anbar, Ramadi 31001, Iraq; co.esamtaha@uoanbar.edu.iq
[2] College of Technical Engineering, University of Al Maarif, Al Anbar 31001, Iraq; sufyan.faraj@uoa.edu.iq
[3] School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool L3 5AH, UK; o.a.aldhaibani@ljmu.ac.uk
* Correspondence: hadeel.saeed@uoanbar.edu.iq

Abstract

A reinforcement learning hyper-heuristic framework for multi-objective scientific workflow scheduling selects among five meta-heuristic optimisers and tunes their control parameters under a Nash Social Welfare reward over makespan, cost, security, and resource utilisation. In its base form it treats security as a static virtual-machine attribute and admits all candidates unconditionally. This paper contributes the mechanism design required to integrate two security-realism layers into the scheduling loop without redesigning the reward: a five-stage zero-trust admission pipeline, a bounded non-stationary per-machine dynamic trust signal, a state-vector augmentation that exposes trust to the agent, and a coupling that attenuates the effective security level seen by the security utility. The two layers act at distinct timescales: admission is a provisioning-time gate on a machine's structural compliance, whereas the trust signal evolves per decision epoch for the machines already admitted, so static admission and dynamic trust coexist by construction. We evaluate three hyper-heuristic agents on 20 Pegasus workflow instances under both a trust suite and a trust-free baseline. The base framework establishes a sharp separation between the hyper-heuristic and direct task-to-machine RL families; we treat this as an inherited property and ask a different question: can the two security-realism layers be integrated without disturbing it? Across 20 Pegasus instances and three HH-RL agents, the family-level separation is preserved. Under a reproducible evaluation protocol—five independently seeded repeats of the full paired comparison, 100 greedy inference episodes per (agent, workflow, suite) cell, with per-workflow deltas averaged across repeats before testing—the trust extension imposes a small, heterogeneous absorption cost: the median per-workflow shift in Nash reward is -0.24, -0.24, and -0.05 for PDQN, DQNHH, and QLHH respectively, an order of magnitude below the absolute reward levels. The shift is statistically significant for DQNHH (two-sided Wilcoxon $p = 0.0014$, rank-biserial r = -0.77), marginal for PDQN ($p = 0.058$), and absent for QLHH ($p = 0.18$). The security utility stays above 0.91 on every instance, and the family-level scaling robustness is preserved intact. The contribution is therefore a drop-in mechanism whose cost is bounded and small relative to the between-family separation—with a robust workflow-level heterogeneity: the parameterised agent converts the trust signal into consistent gains on the largest DAGs (mean +1.28 on Sipht_1000 and Inspiral_1000 across the five repeats) while paying a small cost on typical instances.

Academic Editor: Phivos Mylonas

Received: 9 July 2026
Revised: 29 July 2026
Accepted: 30 July 2026
Published: 5 August 2026

Keywords: reinforcement learning; hyper-heuristic; zero-trust security; dynamic trust; scientific workflow scheduling

1. Introduction

Scientific workflows are increasingly executed on multi-tenant cloud infrastructures whose security guarantees can no longer be assumed by default. Scheduling a workflow under such conditions consists not only of mapping each computational task to a Virtual Machine (VM) under conventional Quality-of-Service constraints—makespan, monetary cost, and resource utilisation—but also of maintaining security compliance during execution and adapting to changes in VM trustworthiness driven by patch state, runtime integrity, load, and policy drift [1–3]. Two complementary security paradigms have emerged in response. The first is the zero-trust model, which mandates that no resource is implicitly trusted and every candidate must pass an explicit verification pipeline before admission. The second is the dynamic trust model, which represents each VM's trustworthiness as a time-varying signal that evolves with observed behaviour. The two are complementary in time as well as in function, and it is worth stating up front that they do not conflict: admission is a one-off, provisioning-time gate on whether a machine is structurally compliant, whereas dynamic trust is a per-epoch signal maintained only for machines that have already been admitted. A fixed admission outcome and a continuously evolving trust value therefore coexist by construction rather than contradicting one another; we make the timescale separation explicit in Section 3.2. In the cloud access-control literature, these two paradigms have been combined to good effect: Wang et al. [4] demonstrate that a zero-trust admission filter together with a deep-RL-controlled dynamic trust threshold yields a substantially more robust access-control regime than either mechanism alone. Crucially, however, that combination has not been transplanted into the workflow-scheduling loop—the scheduler is upstream of access control in any realistic deployment, and its design has so far ignored the zero-trust/dynamic-trust pair.

This paper proposes precisely that transplantation. We build on a Reinforcement Learning Hyper-Heuristic (RL-HH) framework for multi-objective scientific workflow scheduling—referred to throughout as the *base framework*—in which an RL agent observes a continuous level state, selects one of five complete meta-heuristic optimisers (Particle Swarm Optimisation, Simulated Annealing, Tabu Search, Iterated Local Search, and a Randomised Greedy heuristic), and optionally tunes a continuous control parameter before delegating task-to-VM assignment to the chosen optimiser. A Nash Social Welfare reward over four normalised utilities—makespan, cost, security, and utilisation—supplies the training signal. Evaluated here under the trust-free baseline, the base framework exhibits a result that is unusually clean for this literature: the hyper-heuristic family is the *only* learning family that maintains a positive Nash reward at every workflow size across the 20 Pegasus workflow instances. Direct-RL agents—which map tasks to VMs end-to-end without a hyper-heuristic intermediary—collapse to the disagreement-shifted reward floor on the larger DAGs, whereas every hyper-heuristic agent stays in the positive Nash region across the entire size range, with the Parameterised Deep Q-Network (PDQN) variant achieving the strongest individual performance on the largest DAGs. This between-family separation is the dominant empirical feature of the base framework's results, and any extension of the framework has to preserve it to be useful. The base framework, however, models security as a static attribute of each VM: a fixed security provision level is assigned at simulator start-up and is used unchanged thereafter to penalise tasks whose security requirement exceeds it. Two realistic aspects of cloud security are absent from this formula-

tion. First, no admission control is performed, so every VM is implicitly trusted to enter the candidate pool regardless of policy compliance, runtime integrity, or configuration consistency. Second, the trust associated with a given VM does not evolve over time, so an RL agent has no way to learn behaviour that adapts to changing trust conditions.

The principal contribution of this paper is the mechanism design required to integrate zero-trust admission and dynamic trust into the RL-HH scheduling loop without modifying the Nash reward functional. Concretely, the integration introduces four components. A five-stage zero-trust verification pipeline filters candidate VMs through sequential admission checks—identity and registration, policy compliance, secure execution and encryption, security-level consistency, and runtime integrity—so that the RL agent operates only over a verified VM substrate, with each stage abstracting a deployment-time enforcement mechanism (e.g., a registry lookup for stage one; configuration attestation for stage two; an enclave or VM-encryption check such as AMD SEV, Intel TDX, or a TLS/AES envelope on persistent storage and inter-VM traffic for stage three; remote attestation or measured-boot verification for stages four and five). A bounded, non-stationary dynamic trust signal is then maintained for every admitted VM, modelled as an exponentially weighted moving average of an observed trust estimate that can be anchored to the VM's intrinsic execution reliability with an additional random-walk innovation, capturing the unmodelled drift in cloud security state. A state-vector augmentation exposes the per-VM trust vector to the RL agent at every decision epoch, extending the level state from the base framework's dimensionality to one in which the agent observes both the conventional level features and the current trust condition of every candidate VM. Finally, a trust-attenuated effective security coupling propagates the dynamic trust value into the security utility used by the Nash reward, so that a VM nominally provisioned at a high security level but currently carrying a low trust value presents an attenuated effective security level to the security utility; this is the only point at which trust re-enters the reward functional, and we make the coupling explicit rather than treating trust as an inert side channel. The Nash Social Welfare reward, including its weights, disagreement points, and shift, is inherited unchanged from the base framework; the level-based decomposition, the meta-heuristic library, and the agent architectures are inherited unchanged; and the only structural modifications introduced are the four items above. An ancillary contribution is the choice to ground the observed trust estimate in the simulator's existing VM-reliability quantity, which converts the trust update from a free stochastic process into a partially data-driven one parameterised by a single anchoring coefficient.

We validate the mechanism with a controlled experiment on the 20 Pegasus workflow instances that define the base framework's evaluation: five workflow types (Montage, CyberShake, Epigenomics, Inspiral, and Sipht) at four sizes per type. For each of three RL hyper-heuristic agents (PDQN, QLHH, and DQNHH), we train one model under the trust-augmented suite and one under the trust-free baseline, so that any difference is attributable to the mechanism rather than to a single shared policy. The empirical findings provide three pieces of evidence in support of the mechanism design. First, the hyper-heuristic family's superiority over the direct task-to-VM family—established under the trust-free baseline—survives the trust extension cleanly: under the trust suite, the HH-RL family returns a positive Nash reward on 60 of 60 (agent, workflow) cells, whereas the Direct-RL family returns a positive reward on only 15 of 80 cells, with the gap widening sharply on the largest DAGs where the Direct-RL agents collapse to the reward floor while HH-RL agents remain firmly positive. Second, the trust extension does not destabilise the HH-RL family's scaling robustness: HH-RL remains uniformly positive across all four size buckets under the trust suite, while the Direct-RL family remains uniformly collapsed, exactly mirroring the no-trust pattern of the base framework. Third, the security utility under the trust layer

stays above 0.91 on every (agent, workflow) cell, with a median above 0.95, indicating that the structural Nash-floor mechanism interacts safely with the trust-attenuated security coupling. Under a reproducible five-repeat evaluation protocol, the trust layer imposes a small, bounded absorption cost that is statistically significant for DQNHH (two-sided Wilcoxon $p = 0.0014$), marginal for PDQN ($p = 0.058$), and not significant for QLHH ($p = 0.18$), with median per-workflow shifts an order of magnitude below the absolute reward levels; we discuss the implications in Section 4.3. The closest published method to the framework presented here is that of Bajaher et al. [5], who couple a graph-attention state predictor with a DQN-based hyper-heuristic for deadline- and budget-aware workflow scheduling. Bajaher et al. establish empirically that an HH-RL scheduler benefits from a richer state representation than the raw level features; our paper adopts an orthogonal state enrichment—the per-VM dynamic trust vector—and targets a different multi-objective formulation (Nash Social Welfare over makespan, cost, security, and utilisation rather than deadline–budget). Wang et al. [4] established the zero-trust/dynamic-trust/DQN pattern in the cloud access-control setting; our paper transplants the pattern to workflow scheduling and integrates it with the Nash-rewarded RL-HH machinery. To the best of our knowledge, no prior published method combines all three threads: a zero-trust admission layer, a non-stationary dynamic trust signal, and an RL hyper-heuristic scheduler under a multi-objective Nash reward.

The remainder of this paper is organised as follows. Section 2 positions the contribution within the related work on security-aware workflow scheduling, RL hyper-heuristics for workflow scheduling, and zero-trust/dynamic-trust models for the cloud. Section 3 formalises the four-piece trust extension on top of the base framework, with particular attention to where the dynamic trust value actually enters the optimisation. Section 4 reports the empirical evaluation across 20 workflows, three agents, and two suites (trust, no-trust). Section 5 concludes and outlines limitations and future work.

2. Related Work

The contribution of this paper sits at the intersection of three threads in the cloud-scheduling literature: security-and-trust-aware scientific workflow scheduling; reinforcement learning and hyper-heuristic approaches to workflow scheduling; and zero-trust and dynamic-trust models for cloud-resource access control. We review the relevant work in a single sweep, organised by thread but presented in continuous prose, and conclude with a positioning paragraph that identifies the specific gap addressed here.

Classical workflow-scheduling formulations targeted makespan and execution cost as the primary objectives. The body of work that explicitly incorporates security treats it either as a hard feasibility constraint or as an additional optimisation objective alongside the conventional QoS metrics. Li et al. [1] formulate workflow scheduling on heterogeneous cloud VMs as a constrained cost minimisation problem that simultaneously bounds makespan and respects per-task security requirements; their approach treats security as a feasibility filter on VM eligibility rather than as a learnt quantity. In the same spirit but in a hybrid-cloud setting, Abdi et al. [6] develop a deadline-constrained security-aware scheduler that minimises monetary cost via a mathematical programming formulation, with security policy compliance enforced as constraints on inter-cloud data flows. Banerjee and Tekawade [7] extend the cost–reliability trade-off across multiple clouds with explicit security constraints, again using classical optimisation rather than learning. A complementary line of work models security as a soft metric: Sujana et al. [8] embed security into VM selection via fuzzy inference, with the VM choice balancing security demand against makespan; Reddy and Phani Kumar [9] couple a hybrid meta-heuristic with multi-objective security-aware formulations; and Tharani et al. [10] add energy as a third soft objective

alongside security in a fuzzy-rule-driven scheduler. Zade et al. [11] adopt a different stance, treating security as an adversarial cost to be *raised* for would-be attackers, and propose a ring-topology variant of the New Caledonian Crow algorithm to optimise the combined objective. Roy et al. [12] extend security-aware scheduling from a single workflow to multiple concurrent workflows, observing that resource contention itself becomes a security-relevant variable when workflows share VMs. John [13] addresses the dual energy–security trade-off in a mobile-cloud setting via the SEETS framework. These works broaden the operational envelope but retain the assumption that security is either a static VM attribute or a fuzzy-aggregated score. Beyond offline scheduling, El-Kassabi et al. [14] use deep learning for runtime security enforcement in workflow orchestration, adapting policy as anomalies are detected rather than selecting schedulers; this is complementary to the scheduling problem, since enforcement operates at execution time on a schedule that some upstream component has already committed to. Recent surveys [2,3] characterise the space and identify a consistent gap: the literature treats security-aware optimisation, runtime enforcement, and learning-based decision making as separable modules, rarely composed inside a single learning agent.

The application of reinforcement learning to workflow scheduling has progressed from direct task-to-VM mapping agents—a single deep Q-network learning the assignment policy—towards hyper-heuristic agents that select among established meta-heuristic optimisers. Krishna and Mangalampalli [15] propose PWSA3C, an Asynchronous Advantage Actor Critic scheduler that ranks tasks by priority and dependency before dispatching to VMs, reporting improvements over DQN, A2C, and meta-heuristic baselines on Epigenomics and LIGO workflows; this is representative of the direct-RL family, in which the agent's action space is the VM index, the state is task- and VM-level features, and the policy is learned end-to-end. Udomkasemsub et al. [16] introduce PHH, a policy-based hyper-heuristic in which an RL controller selects among low-level meta-heuristics rather than mapping tasks directly; their evaluation spans the travelling salesman, vehicle routing, bin packing, and deadline-constrained workflow scheduling problems, with strong generalisation reported on unseen larger workflow instances. PHH establishes the empirical value of the hyper-heuristic action space; it does not consider security or trust, and its reward is a deadline-aware scalar rather than a multi-objective Nash aggregate. The closest published method to ours is that of Bajaher et al. [5], who couple a Multihead Graph Attention Network (MGAN) for state prediction with a DQN-based hyper-heuristic for deadline- and budget-aware workflow scheduling: their MGAN provides a learned, structured representation of inter-task and task–resource relations from which the DQN selects a low-level heuristic. Bajaher et al. thus already establish that a hyper-heuristic RL agent over scientific workflows benefits from a richer state representation than the raw level features. Their focus, however, is on deadline–budget trade-offs; security and trust are not modelled, and the four-objective Nash aggregation used by our base framework is not part of their formulation. Multi-objective PSO variants such as MOHIPSO [17] demonstrate that classical swarm-intelligence schedulers can balance makespan, execution time, and resource utilisation, and they remain useful baselines; where RL approaches dominate is in the ability to adapt the choice of optimiser to the workflow state at decision time—the core idea of the hyper-heuristic framing.

Beyond workflow scheduling proper, a vigorous recent line of work applies deep reinforcement learning to adjacent cloud–edge resource-management problems, and it motivates the learning-based stance we adopt here. Chen et al. [18] combine personalised federated learning with DRL for joint computation offloading and resource allocation across multi-edge smart communities; the same line of work extends robust federated learning to resilient collaborative caching in multi-edge systems [19], applies DRL to intelligent

offloading in blockchain-based mobile crowdsensing [20], and couples multi-agent RL with hierarchical knowledge transfer for joint service caching and resource allocation in digital-twin-empowered cloud–edge networks [21]. These works target offloading and caching rather than security-aware DAG scheduling, but they corroborate two premises of our design: that deep RL is an effective controller for heterogeneous cloud–edge resource allocation, and that robustness and trust—federated-learning resilience and blockchain-backed auditability—are increasingly first-class concerns in that setting, concerns that our zero-trust and dynamic-trust layer brings directly into the scheduling loop.

The trust mechanisms used in this paper draw on a body of work in cloud access control rather than in scheduling. The most directly relevant prior work is that of Wang et al. [4], who propose a trust-based access-control model (TBAC) inspired by Zero-Trust Architecture and a dynamic refinement (DR-TBAC) that uses a DQN to update trust thresholds and adapt access policies over time. Their setting is access control rather than workflow scheduling—the RL agent regulates whether and when a principal is admitted to a resource, not which tasks should run where—but the structural pattern is the same one we use: zero-trust as an admission filter, dynamic trust as a time-varying signal, and an RL agent operating on top. In addition to the trust-state signal, Wang et al. rely on standard cloud encryption primitives—transport-layer security for inter-component traffic and authenticated symmetric encryption for persistent state—as the substrate against which the verification checks are defined, and confidential-computing primitives such as AMD SEV, Intel TDX, and Trusted Execution Environments provide the runtime integrity guarantees that make attestation-style checks meaningful. Our paper adapts that pattern to the workflow scheduling problem, integrates it with a multi-objective Nash reward, and assumes the same class of underlying enforcement mechanisms—in particular, the secure-execution check in our verification pipeline is intended to correspond to an enclave or VM-encryption status (e.g., a SEV/TDX guest with attested disk and network encryption), although the pipeline itself is abstract over the specific enforcement choice. Dorsala et al. [22] survey blockchain-based mechanisms for adding decentralised trust and auditability to cloud services; blockchain is one possible realisation of the identity-registration and security-level-consistency checks in our zero-trust pipeline, with hash-chained attestation records playing the role of an audit log over the registry entries. He et al. [23] report a deep-RL real-time scheduler for hybrid clouds with privacy- and security-aware modelling; this is the closest existing work that combines deep RL with security-aware scheduling, but the agent acts directly on the task-to-VM mapping and the trust mechanism is not the focus. We share with He et al. the broad goal of learning under security-aware multi-objective rewards, and differ in (i) the hyper-heuristic action space, (ii) the explicit zero-trust admission layer, and (iii) the use of a Nash Social Welfare aggregation.

Three observations follow from the review above. First, the security-aware scheduling literature is rich but largely treats security as a static or fuzzy VM attribute; the non-stationary character of cloud trust is rarely modelled inside the scheduling loop. Second, the RL-for-workflow literature has converged on the hyper-heuristic action space as the way to extract scaling robustness on large DAGs, but has so far focused on cost–deadline objectives without admitting security as a first-class concern. Third, the zero-trust and dynamic-trust literature has produced machinery for access control—including the standard encryption primitives (transport-layer security, authenticated-encryption at rest) and confidential-computing technologies (SEV, TDX, TEEs) that the verification stages presuppose—that has not yet been transplanted into the scheduling loop. The paper presented here sits in the intersection that no previous work occupies: a zero-trust admission layer plus a non-stationary dynamic trust signal, integrated into a multi-objective Nash-rewarded RL hyper-heuristic scheduler, with the trust signal coupling through both the

agent's state vector and a multiplicative attenuation of the effective VM security level seen by the security utility. The remainder of the paper formalises this integration (Section 3) and reports an empirical evaluation (Section 4) on 20 Pegasus workflow instances. Table 1 lists the comparison dimensions used in Table A1, where each related work is scored along the standard taxonomy used by recent surveys.

Table 1. Abbreviations used in the literature comparison table (Appendix A).

Abbrev.	Description
T&C	Explicit optimisation of execution time (makespan) and monetary cost.
En	Energy consumption/power efficiency considered explicitly.
Sec	Explicit modelling of security, privacy, risk, trust, or attack-related factors.
B/D	Budget and/or deadline constraints (SLA-like).
MH	Meta-heuristic optimisation (PSO/GA/ACO/GWO/SA, etc.).
RL/HH	Reinforcement learning and/or hyper-heuristic.
Sel-MH	Selection among complete meta-heuristic optimisers (not just parameter tuning).
Nash	Nash/game-theoretic utility formulation for multi-objective balancing.
ZT	Zero-trust admission/verification layer for VMs.
DynT	Dynamic, time-varying trust signal modelled in the scheduler.

3. Methodology

The objective of this paper is to design and integrate two security-realism mechanisms—a zero-trust verification pipeline and a dynamic per-VM trust signal—on top of the base RL hyper-heuristic framework for multi-objective scientific workflow scheduling. Throughout, we treat the underlying RL-HH framework as a fixed substrate: the agents, the hyper-heuristic action space, the meta-heuristic library, and the Nash Social Welfare reward functional are inherited without modification. This design choice is deliberate. The base framework establishes that the hyper-heuristic RL family is the only learning family in the comparison that delivers positive Nash reward at every workflow size, with the direct task-to-VM RL family collapsing to the reward floor on the larger DAGs; preserving that family-level superiority is a binding constraint on the trust extension, and our composition is engineered specifically so that it slots in without disturbing the components that produce it. The contribution of this section is the *mechanism design* that couples zero-trust admission control and dynamic trust to that substrate, together with an explicit account of how the dynamic trust value propagates through the existing security utility.

A non-trivial design decision concerns where trust enters the optimisation problem. We do *not* introduce trust as a new objective in the Nash reward; the four-objective Nash functional (makespan, cost, security, utilisation) is left intact. Trust enters in two specific places: (i) as an admission-control filter that shrinks the candidate VM set prior to scheduling, and (ii) as a multiplicative attenuation of the *effective* security level used in the security utility u_{sec}. The second of these two couplings means that trust is not a purely contextual signal: low trust on a chosen VM degrades u_{sec} and therefore the Nash reward. This is an intentional design choice—security guarantees should only be as strong as the trust the substrate currently warrants—and we make it explicit here rather than treating trust as an inert side channel.

3.1. Workflow and Cloud Model

A scientific workflow is represented as a Directed Acyclic Graph $\mathcal{W} = (\mathcal{T}, \mathcal{E})$, where $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$ is the set of workflow tasks and $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ denotes precedence constraints. An edge $(t_i, t_j) \in \mathcal{E}$ indicates that t_j cannot start before t_i has completed. Each task t_i is characterised by

$$t_i = \big(\text{length}(t_i),\ \text{file\_in}(t_i),\ \text{file\_out}(t_i),\ \text{req\_sec}(t_i)\big),$$

with req_sec$(t_i) \in \{0, 1, 2\}$ corresponding to low, medium, and high security requirements, respectively. Here the argument t_i indexes a task, distinct from the decision epoch t used in the trust signal $T_v(t)$.

The cloud environment consists of K heterogeneous VMs $\mathcal{V} = \{v_1, \ldots, v_K\}$. Each VM is defined as

$$v = \big(MIPS_v,\ C_v,\ Sec_v,\ R_v,\ T_v(t)\big),$$

where $MIPS_v$ is the processing capacity, C_v is the per-second execution cost rate, $Sec_v \in \{0, 1, 2\}$ is the nominal security level provisioned for the VM, $R_v \in (0, 1]$ is the VM execution reliability used by the simulator, and $T_v(t) \in [0, 1]$ is the dynamic trust value at decision epoch t. The first four attributes are static across an episode; $T_v(t)$ is the only time-varying VM attribute.

The fleet size K is set automatically as $K = \max(1, \lfloor 1.4\sqrt{N_{\text{tasks}}} \rfloor)$, as in the base framework. Per-VM cost rates scale with $MIPS_v$ so that faster VMs are also more expensive, preventing the optimiser from collapsing onto a single ultra-fast machine.

3.2. Zero-Trust Verification Layer

The first of the two trust mechanisms is a zero-trust admission-control layer interposed between fleet instantiation and scheduling. No VM is implicitly trusted to participate in scheduling, regardless of registration status; every candidate must pass an explicit verification pipeline before being admitted to the candidate pool.

We define a verification function $\Gamma : \mathcal{V} \to \{0, 1\}$ as a product of five binary checks,

$$\Gamma(v) = \prod_{m=1}^{5} \gamma_m(v), \qquad \gamma_m(v) \in \{0, 1\}. \tag{1}$$

A VM is admitted only when all five checks pass. The five checks are summarised in Table 2, which names both the binary flag maintained on each VM in the simulator implementation *and* the cryptographic or enforcement mechanism that the stage is intended to abstract in a real deployment.

Table 2. Five-stage zero-trust verification pipeline. Each stage maps to a binary VM flag set at fleet instantiation and corresponds to a concrete cryptographic or enforcement mechanism in a real deployment. A VM is admitted to the candidate set $\mathcal{V}^{ZT}$ only when all five flags evaluate to 1. The simulator abstracts the enforcement mechanisms as binary flags; the third column names the mechanism that each flag stands in for.

Stage	Check	Implementation Flag	Encryption/Enforcement Substrate
1	VM identity and registration validity	`is_registered`	Mutual TLS (mTLS) registration with a cloud-provider public-key infrastructure (PKI); X.509 certificate pinning; optionally a blockchain-anchored identity registry.
2	Configuration and policy compliance	`is_policy_compliant`	Signed configuration manifests verified against a policy engine (e.g., OPA/Rego); image-digest verification (Docker Content Trust/cosign/Sigstore) over the VM image and its bootstrap artefacts.
3	Secure execution and encryption	`is_secure_exec`	Confidential-computing runtime (AMD SEV-SNP, Intel TDX, or ARM CCA) for memory-encrypted execution; LUKS/dm-crypt AES-XTS for data-at-rest; TLS 1.3 (or QUIC) with authenticated AEAD ciphers (AES-GCM or ChaCha20-Poly1305) for data-in-transit between VMs and to storage.

Table 2. *Cont.*

Stage	Check	Implementation Flag	Encryption/Enforcement Substrate
4	Security level consistency	`sec_level_consistent`	Signed security-level attestation from the cloud control plane; HMAC-bound label metadata; cross-checked against the provider's KMS-signed inventory record so that the advertised Sec_v cannot be tampered with.
5	Runtime integrity and state consistency	`is_runtime_integral`	TPM-based measured boot with remote attestation (TPM 2.0 quotes, IMA/EVM event logs); periodic re-attestation of the running kernel and critical user-space modules; verified-boot chain rooted in a hardware Root of Trust.

The set of admitted candidates is

$$\mathcal{V}^{ZT} = \{\, v \in \mathcal{V} \mid \Gamma(v) = 1 \,\}. \tag{2}$$

The scheduling mapping is then refined from the unconstrained form $\phi : \mathcal{T} \to \mathcal{V}$ to

$$\phi : \mathcal{T} \to \mathcal{V}^{ZT}, \tag{3}$$

where we write the task-to-VM assignment as ϕ to avoid collision with the trust-walk standard deviation σ of Equation (5). Only VMs in $\mathcal{V}^{ZT}$ are exposed to the RL agent and to the meta-heuristic optimisers it dispatches.

3.2.1. Failure Model

At fleet instantiation each VM is independently subjected to the pipeline with a per-VM failure probability $\rho_{ZT} \in [0, 1]$ on a uniformly chosen single check. In all experiments reported in Section 4, $\rho_{ZT} = 0.10$. As a fallback, if no VM passes verification on a particular episode the simulator defers to the full fleet rather than halting, so that training data is never lost to a pathological draw; in practice this fallback was never triggered.

3.2.2. Static Admission Versus Dynamic Trust

The two mechanisms play deliberately different temporal roles, and separating them resolves an apparent tension between a fixed admission draw and the "dynamic trust" claim. Zero-trust admission is a *static*, provisioning-time gate: it models whether a VM is structurally compliant—valid registration, attested configuration, and consistent security labelling—at fleet instantiation, and a VM that fails such a check is appropriately excluded for the episode, since structural compliance is not a property that should fluctuate from level to level. The *dynamic* element of the framework is the trust signal $T_v(t)$, which evolves over time for the *admitted* VMs and is the quantity through which time-varying confidence enters both the state (Coupling 2) and the security utility (Coupling 3). The "dynamic trust" claim thus refers to $T_v(t)$, not to the admission gate, and the two are not in conflict. Two further points follow. First, the full-fleet fallback is a defensive environment invariant, not a learned policy branch: it merely prevents an episode from presenting an empty action set, and because an empty verified set has probability $\approx \rho_{ZT}^{K}$ it remains essentially never triggered even at the elevated $\rho_{ZT} = 0.30$ of the sensitivity sweep; in a real deployment, an empty verified set would raise an operational alarm rather than schedule silently. Second, in the present study, the verification outcome is drawn once per fleet seed and held fixed across training episodes, so the agent trains against a fixed admitted subset. The multi-seed evaluation of Section 4.6 varies this subset across five seeds and finds the

learned policy robust to which VMs are excluded; resampling the verification outcome *per episode* during training—a stochastic-admission variant closer to a continuously re-verified deployment—is a natural extension we leave to future work.

3.2.3. Scope of the Model

The pipeline is deliberately stylised: each γ_m is a binary flag rather than a substantive runtime check. The contribution claimed here is not the verification logic of any single γ_m in isolation but the *integration* of admission control into the RL-HH scheduling loop: the agent learns over a verified VM substrate rather than over the raw fleet. Substituting any specific γ_m with a richer mechanism (e.g., a full TPM-2.0 remote-attestation backend for stage 5, or a provider-signed configuration manifest verified through a Rego policy engine for stage 2) leaves the rest of the framework unchanged.

3.2.4. Encryption Substrate

Although the simulator treats each γ_m as a binary flag, every stage of the pipeline presupposes a concrete cryptographic substrate, and we name it here so that the abstraction is reproducible. Identity registration (stage 1) is mediated by mTLS with X.509 certificates issued by a cloud-provider PKI. Configuration and policy compliance (stage 2) relies on signed image digests—specifically the Docker Content Trust/Sigstore/cosign chain—and on policy decisions emitted by an OPA/Rego policy engine. Secure execution (stage 3) is realised by a confidential computing runtime that encrypts VM memory at the hardware level: AMD SEV-SNP, Intel TDX, or ARM CCA are the three mainstream options; data at rest is protected by LUKS/dm-crypt with AES-XTS, and data in transit by TLS 1.3 with an authenticated-encryption AEAD cipher such as AES-GCM or ChaCha20-Poly1305. Security-level consistency (stage 4) is verified by checking an HMAC-bound, KMS-signed label emitted by the provider's control plane against the locally advertised Sec_v. Runtime integrity (stage 5) rests on TPM-2.0 measured boot with remote attestation quotes (IMA/EVM event logs anchored at the TPM PCRs) and a verified-boot chain whose Root of Trust is a hardware element on the host. These mechanisms are not novel contributions of this paper, and our simulator does not implement them; naming them explicitly is intended to make the verification pipeline concrete enough that a real deployment can map each γ_m to a deterministic enforcement check, and to clarify that the trust-attenuation coupling of Equation (7) sits on top of a fully cryptographically protected substrate rather than in place of one.

3.3. Dynamic Trust Modelling

The second mechanism is a per-VM dynamic trust signal $T_v(t) \in [0,1]$ maintained only for VMs in $\mathcal{V}^{ZT}$. We model $T_v(t)$ as an exponentially weighted moving average (EMA) of an observed trust estimate $\widehat{T}_v(t)$, with the EMA acting as a low-pass filter on the underlying observation:

$$T_v(t+1) = (1-\alpha)\, T_v(t) + \alpha\, \widehat{T}_v(t), \qquad v \in \mathcal{V}^{ZT}, \tag{4}$$

with $T_v(t) \in [0,1]$ clipped after each update and $\alpha \in (0,1]$ the trust adaptation coefficient. At the start of each episode, $T_v(0) = 1.0$ for every verified VM.

Observation Model

The observed trust estimate $\widehat{T}_v(t)$ is intended as a coarse proxy for the unobserved security state of the VM (patch currency, runtime integrity, anomalous load, policy drift). Rather than treating $\widehat{T}_v$ as a free stochastic process disconnected from the simulator state, we ground it in two quantities that the simulator already maintains: the VM execution

reliability R_v and a random-walk innovation that represents unmodelled drift. Concretely, $\widehat{T}_v(t)$ evolves as

$$\widehat{T}_v(t+1) = \mathrm{clip}_{[0,1]}\Big(\beta\, R_v + (1-\beta)\,\widehat{T}_v(t) + \eta_v(t)\Big), \qquad \eta_v(t) \sim \mathcal{N}(0, \sigma^2), \tag{5}$$

with $\beta \in [0,1]$ controlling how strongly observed trust is anchored to the VM's intrinsic reliability and σ the standard deviation of the unmodelled-drift component. In the experiments reported in Section 4 we use $\beta = 0$ throughout, so $\widehat{T}_v$ reduces to a clipped Gaussian random walk and the trust dynamics are fully exogenous; reporting β explicitly makes this an experimental choice rather than a hidden assumption.

We choose $\beta = 0$ deliberately, and state its meaning precisely so it is not mistaken for a shortcoming: at $\beta = 0$ the observed estimate is not anchored to any measured VM behaviour, so the trust signal in the reported experiments is a *controlled exogenous stimulus* injected into the scheduler rather than the output of any particular trust estimator. This is a deliberate experimental-design choice, not a limitation of the mechanism. A grounded estimator is fully supported by the model—it is what $\beta > 0$ activates—but coupling one in for the main experiments would confound the object of study. Our object of study is how the RL-HH machinery *responds* to a bounded, non-stationary trust signal once that signal is coupled into the state and the security utility; setting $\beta = 0$ isolates that response from the confounding influence of a particular observation model, so that an observed reward shift cannot be attributed to the cleverness (or brittleness) of a trust estimator. Equation (5) is written in the general anchored form precisely to expose the hook through which a grounded estimate enters: with $\beta > 0$ the observation update anchors to the VM execution reliability R_v, and replacing the R_v term with a telemetry- or anomaly-detection-driven score (integrity attestation, side-channel signals, learned anomaly scores) would make $\widehat{T}_v$ genuinely behaviour-grounded. Implementing that anchored observation update and sweeping β is a direct extension of Equation (5) rather than a redesign of the scheduler, and we identify it as the primary line of future work (Section 5).

Equation (4) produces a bounded, non-stationary trust signal that the scheduler can observe between workflow levels. Trust evolution is invoked once per level, immediately after the level's tasks have been dispatched and executed, so that within-level scheduling decisions operate on a fixed trust snapshot.

3.4. How Trust Enters the Optimisation

It is important to be precise about where the trust signal actually affects training. Three coupling points exist in the implementation, and each is a deliberate design choice that referees should be able to inspect independently.

3.4.1. Coupling 1: Admission Control

Through Equation (2) the candidate set is shrunk from $\mathcal{V}$ to $\mathcal{V}^{ZT}$. When $\rho_{ZT} = 0.10$ and the fleet has K VMs, the expected admitted fleet is $0.9\,K$. This is a discrete-feasibility modification of the action space; the Nash reward functional is unchanged.

3.4.2. Coupling 2: State Augmentation

The state vector observed by the RL agent at the start of each workflow level is extended from the base framework's $8 + 2K$ dimensions to $8 + 3K$ dimensions by appending the trust vector $\mathbf{T}_t = [T_{v_1}(t), \ldots, T_{v_K}(t)]$. Letting s_t denote the base framework's level state, the augmented state is

$$s'_t = [s_t,\ \mathbf{T}_t]. \tag{6}$$

This gives the agent direct visibility into the per-VM trust condition before it commits to a scheduling heuristic and its control parameter. The agent may or may not exploit this extra dimension; whether it does is an empirical question addressed in Section 4. We do not claim novelty for the state-augmentation operation itself: appending a per-VM trust vector to the state is a standard augmentation, and we present it as such. The contribution is not this single step but the *composition* of admission control (Coupling 1), the exposure of trust to the agent (Coupling 2), and the trust-attenuated security utility (Coupling 3) into a drop-in extension that preserves the base framework's reward functional and scaling behaviour—a point we return to in Section 3.11.

3.4.3. Coupling 3: Trust-Attenuated Effective Security

Inside the simulator, the security utility u_{sec} that feeds the Nash reward is computed not from the nominal VM security level Sec_v but from a *trust-attenuated effective* level

$$Sec_v^{\text{eff}}(t) = \lfloor Sec_v \cdot T_v(t) \rfloor, \tag{7}$$

which is then compared to each task's req_sec(t_i) when accumulating security violations. Concretely, a VM nominally provisioned at $Sec_v = 2$ but currently carrying $T_v(t) = 0.4$ presents an effective security level of $\lfloor 0.8 \rfloor = 0$ to the security utility, and tasks assigned to it with req_sec > 0 incur a security violation. This is the only place in the framework where the dynamic trust value re-enters a scalar that is summed into the Nash reward.

We make this third coupling explicit because it has a practical implication for interpretation. Any reward differential observed between the trust and no-trust suites in Section 4 cannot be attributed purely to state-space dimensionality or to admission filtering: it also reflects a structural change in how u_{sec} is computed. The Nash reward functional itself is unchanged, but the inputs that feed one of its four utility terms are.

Why a Multiplicative Floor

The coupling in Equation (7) is a deliberate design choice, and we justify its two components explicitly. The *multiplicative* form treats trust as a proportional derating of the security capability a VM actually delivers: at full trust $T_v(t) = 1$ the effective level equals the nominal level (identity), and as trust erodes the effective level scales down proportionally, reaching zero only when trust collapses. This "capability $\times$ confidence" semantics is precisely the property we want—trust modulates the provisioned security rather than adding to or subtracting from it, so it can never credit a VM with more security than it was provisioned and requires no artificial clamping. An additive alternative such as $Sec_v - \kappa\,(1 - T_v)$ would need both an extra gain κ and clipping to remain in range, and would not preserve the $T_v = 1$ identity cleanly. The *floor* maps the continuous product back onto the discrete ordinal domain $\{0, 1, 2\}$ shared by Sec_v and each task's req_sec(t_i), so that the effective level can be compared against task requirements by the base framework's existing integer violation rule $\max\big(0,\ \text{req\_sec}(t_i) - Sec_v^{\text{eff}}\big)$ without redefining the security utility. Among the possible discretisations, the floor is the conservative, fail-safe choice consistent with the zero-trust stance: under uncertainty it rounds the effective security *down*, so trust erosion can only ever tighten, never relax, the security check.

Relation to Alternative Couplings

Two alternatives are worth naming. A *continuous-decay* coupling $Sec_v^{\text{eff}}(t) = Sec_v \cdot T_v(t)$ omits the floor and keeps a real-valued effective level; it degrades security more gently (always partial credit) but requires the security utility and its violation rule to be redefined over a continuous domain, breaking the drop-in compatibility with the base framework that is the point of this paper. A *threshold-gating* coupling $Sec_v^{\text{eff}}(t) = Sec_v\,\mathbf{1}[T_v(t) \geq \tau_{\text{th}}]$

collapses the effective level to zero once trust falls below a cutoff τ_{th}; this introduces a free threshold and a hard discontinuity that can destabilise learning, and is in effect a coarser, single-step special case of the multi-step degradation that $\lfloor Sec_v \cdot T_v(t) \rfloor$ already provides across the ordinal levels. Our floor coupling is thus the discretisation-consistent middle ground between these two. A systematic empirical comparison of the three couplings is a natural ablation; because the sensitivity analysis of Section 4.6 shows u_{sec} is saturated and robust in the tested regime, we expect the coupling choice to have limited leverage on the reported results and leave a full comparison to future work.

3.5. Multi-Objective Scheduling Formulation

The scheduling problem seeks a mapping $\phi : \mathcal{T} \to \mathcal{V}^{ZT}$ that respects DAG precedence and jointly optimises the four performance criteria inherited from the base framework: makespan, execution cost, security compliance, and resource utilisation. These criteria are aggregated into the normalised utilities u_{time}, u_{cost}, u_{sec}, $u_{\text{res}} \in [0, 1]$ exactly as in the base framework, with one modification: u_{sec} now uses $Sec_v^{\text{eff}}(t)$ in place of Sec_v via Equation (7).

3.6. Level-Based Workflow Decomposition

To bound decision complexity, the DAG is decomposed into execution levels. Tasks at the same level are mutually independent and may execute in parallel once all preceding levels have completed. Let $\mathcal{L} = \{L_1, L_2, \ldots, L_D\}$ denote the ordered set of levels, with L_d the task set at level d. The RL agent operates sequentially over $\mathcal{L}$. At each level, the agent observes the augmented state of Equation (6), selects a scheduling action, delegates task-to-VM assignment to the chosen meta-heuristic, and finally invokes the trust-update step of Equations (4) and (5) between levels. This level granularity is unchanged from the base framework.

3.7. Action Space and Hyper-Heuristic Control

The action at each decision epoch is a hybrid discrete–continuous pair

$$a_t = \left(a_t^d,\ a_t^c\right),$$

where the discrete component $a_t^d \in \{0, 1, 2, 3, 4\}$ selects one of five meta-heuristics from a fixed library, and the continuous component $a_t^c \in [0, 1]$ tunes the chosen meta-heuristic's internal control parameter (denoted θ in the empirical analysis of Section 4). The library comprises Particle Swarm Optimisation (PSO), Simulated Annealing (SANN), Tabu Search (TABU), Iterated Local Search (ILS), and a Randomised Greedy constructor (RG). Each metaheuristic, once selected and parameterised, consumes the current verified VM set $\mathcal{V}^{ZT}$ and the task list L_d and returns a complete task-to-VM assignment for that level.

This hybrid action structure motivates the use of a Parameterised Deep Q-Network (PDQN) as the principal hyper-heuristic agent, since PDQN is designed to learn discrete selection and continuous parameterisation jointly. For completeness the empirical evaluation also includes a tabular agent (QLHH) and a neural fixed-θ agent (DQNHH); these two operate on a discrete-only action space with a_t^c held at a fixed default.

3.8. Reward Function

The terminal Nash Social Welfare reward used to train the agent is identical to the one used by the base framework:

$$r = \lambda \sum_{i \in \{\text{time, cost, sec, res}\}} \omega_i \ln\big(\max(\epsilon,\ u_i - d_i)\big) + \text{shift}, \tag{8}$$

where $\omega_i \geq 0$ with $\sum_i \omega_i = 1$ (the objective weights, denoted ω_i to avoid collision with the workflow index w used in Section 4), $\lambda > 0$ is a scaling factor, $\epsilon > 0$ is a numerical-stability constant, d_i is the disagreement (utility-floor) point of objective i, and shift is a constant additive bias. The functional form, the weights, the disagreement points, and the shift are all inherited from the base framework without change; the only difference under the trust suite is in the input to u_{sec} via Equation (7).

In particular, the disagreement point $d_{\text{sec}} = 0.50$ acts as a hard floor: if the trust-attenuated effective security degrades to the point where $u_{\text{sec}} \to d_{\text{sec}}$, the $\ln(u_{\text{sec}} - d_{\text{sec}})$ term diverges negatively and the episode reward collapses. This makes d_{sec} a structural safety mechanism: the trust-attenuation coupling cannot silently push the agent into low-security solutions because the Nash reward floor penalises it. Whether this safety mechanism is actually exercised in practice is examined empirically in Section 4.4.

3.9. State Representation in Detail

The augmented state $s'_t = [s_t, \mathbf{T}_t]$ of Equation (6) packs the following components, in the order in which they are emitted by the simulator:

- Current workflow level progress, normalised by the DAG's maximum depth;
- Number of tasks in the current level, normalised by the maximum level cardinality across the DAG;
- Average and maximum task length in the current level (normalised);
- Average task security requirement in the current level (normalised);
- Per-VM ready times $\{\tau_v\}_{v=1}^{K}$ (normalised);
- Per-VM processing capacities $\{MIPS_v \cdot R_v\}$ (normalised);
- Current accumulated makespan, cost, and security score (normalised);
- Per-VM dynamic trust values $\mathbf{T}_t = [T_{v_1}(t), \ldots, T_{v_K}(t)]$ (when the trust mechanism is enabled).

The first eight blocks coincide with the base framework's s_t; the trust vector is the only state-space extension introduced by this paper.

3.10. Learning Procedure

The scheduling procedure unfolds episodically. At the start of each episode the workflow DAG is loaded and decomposed into levels; every verified VM has its trust initialised to $T_v(0) = 1$. At each level the environment constructs the augmented state s'_t; the agent (PDQN, QLHH, or DQNHH) selects a discrete heuristic and, in the case of PDQN, a continuous control parameter; the selected meta-heuristic is invoked on the level's task list and the verified candidate set; tasks are dispatched to their assigned VMs; the simulator advances time, accumulates makespan, cost and the trust-attenuated security score; and the trust update of Equations (4) and (5) is applied to every verified VM before the next level begins.

Once all levels have been scheduled and the workflow has completed, the terminal Nash reward of Equation (8) is computed and used to update the RL agent. Over E training episodes the agent learns a scheduling policy that balances the four cloud-allocation objectives, with the trust signal acting both as a state-space cue (Coupling 2) and as a multiplicative attenuation of the effective security level seen by u_{sec} (Coupling 3).

3.11. Summary of Methodological Contribution

The methodological contribution of this paper is the integration of two security-realism mechanisms into the RL-HH scheduling loop without redesigning the Nash reward functional: (i) a five-stage zero-trust pipeline that restricts the action space to a verified VM substrate, and (ii) a bounded, non-stationary dynamic trust signal that augments the state

vector *and* multiplicatively attenuates the effective security level entering the security utility. The framework does not propose a novel trust estimation algorithm; the observation model of Equation (5) is deliberately simple so that the empirical analysis isolates the agent's response to the mechanism rather than the cleverness of the trust estimator. What the framework does propose is a clean, drop-in composition: any RL agent that operates on the base framework's level state can be evaluated on the trust suite by feeding it the augmented state s'_t and the trust-attenuated u_{sec}, with no further changes to its learning loop.

4. Experimental Results

This section evaluates the trust-augmented framework against the trust-free baseline on the 20 Pegasus workflow instances that define the base framework's evaluation. Section 4.1 describes the protocol; the remaining subsections analyse the results from five complementary angles. Two are agent-centric: a per-workflow paired comparison (Section 4.2) and a statistical test of the paired deltas with explicit effect sizes (Section 4.3). Two are mechanism-centric: the behaviour of the security utility under the trust-attenuated coupling (Section 4.4), and the scaling behaviour of HH-RL versus Direct-RL under both suites (Section 4.5). The final subsection (Section 4.7) inspects PDQN's training trajectory under the trust suite to confirm that the inference-time behaviour reflects a stable learning process.

The headline empirical findings of this section are twofold. The dominant finding is that the hyper-heuristic RL family decisively outperforms the direct task-to-VM RL family under the trust suite: HH-RL agents return a positive Nash reward on 60/60 (agent, workflow) cells, whereas Direct-RL agents return a positive reward on only 15/80 cells, and on the largest workflows the gap between the best HH-RL agent and the best Direct-RL agent widens to between 1.16 and 6.60 Nash units—a separation that grows rather than shrinks with workflow size. Layered on top of this between-family superiority is a within-family absorption result: the trust mechanism is absorbed by the HH-RL family at a small, bounded cost that does not disturb its family-level balance. Under a reproducible five-repeat evaluation protocol (Section 4.1), the trust extension imposes a modest downward shift in Nash Total Reward that is statistically significant for one of the three agents (DQNHH, two-sided Wilcoxon $p = 0.0014$), marginal for PDQN ($p = 0.058$), and not significant for QLHH ($p = 0.18$); the median per-workflow shifts of -0.24 to -0.05 Nash units are an order of magnitude below the absolute reward levels, the security utility remains uniformly high, and the family-level scaling story of the base framework survives intact. Together these two findings constitute the empirical evidence in support of the mechanism contribution claimed in Section 3: zero-trust admission and dynamic trust can be composed into the RL-HH scheduling loop without disturbing the scaling robustness that makes the HH-RL family dominate the comparison in the first place.

4.1. Experimental Setup

The simulation environment, VM-fleet sizing, Nash Social Welfare reward, training protocol ($E = 10{,}000$ episodes per agent–workflow pair), and inference protocol (ten greedy inference episodes with no exploration (a purely greedy policy); deterministic baselines reported as single runs) are inherited unchanged from the base framework; we report only the configuration items specific to the trust extension.

Trust suite.

At simulator start-up, each VM is independently subjected to the five-stage zero-trust verification pipeline of Table 2 with a per-VM failure rate of $\rho_{ZT} = 0.10$ on a uniformly chosen single check. VMs that fail verification are excluded from the candidate set $\mathcal{V}^{ZT}$ and never receive task assignments. Every verified VM additionally carries a dynamic trust

signal $T_v(t) \in [0,1]$ initialised at $T_v(0) = 1.0$ and updated between workflow levels via Equation (4) with adaptation coefficient $\alpha = 0.1$ and Gaussian-walk standard deviation $\sigma = 0.05$. The observation anchoring coefficient is held at $\beta = 0$, so the observed trust estimate $\widehat{T}_v(t)$ reduces to a clipped Gaussian random walk; this isolates the agent's response to the mechanism from confounding effects of a structured observation model. The RL state vector is augmented as $s'_t = [s_t, \mathbf{T}_t]$ per Equation (6), and the security utility u_{sec} is computed using the trust-attenuated effective security $Sec_v^{\text{eff}}(t)$ of Equation (7). The Nash reward functional itself is unchanged.

No-Trust suite (baseline).

The baseline disables every component of the trust extension in lock-step: no zero-trust filtering ($\mathcal{V}^{ZT} = \mathcal{V}$), no dynamic trust signal, the state vector is the base framework's s_t (dimension $8 + 2K$), and u_{sec} is computed from the nominal Sec_v. This is the same configuration under which the base framework reports its main results.

Agents evaluated.

The empirical analysis focuses on the three RL hyper-heuristic agents: QLHH (tabular state, fixed θ), DQNHH (neural state, fixed θ), and PDQN (parameterised, learned θ). For each (agent, workflow) cell, a separate model is trained under each suite, so that any difference in behaviour is attributable to the trust mechanism rather than to a single shared policy. Where Direct-RL agents are referenced (Section 4.5) they are reported solely to verify that the family-level scaling behaviour observed under the no-trust suite is preserved under the trust suite.

Evaluation protocol and reproducibility.

The paired comparison is evaluated under a fully seeded, repeated protocol. Each (agent, workflow, suite) cell is evaluated with 100 greedy inference episodes, and the entire paired table is regenerated five times under independent documented environment seeds (42, 142, 242, 342, 442); per-workflow deltas are averaged across the five repeats before statistical testing, so that each workflow contributes exactly one paired observation ($n = 20$ per agent) and no pseudoreplication is introduced. Two sources of run-to-run variability make this protocol necessary rather than cosmetic. First, the trust environment is stochastic even under a deterministic greedy policy: the trust random walk of Equation (5), the zero-trust admission draw, and the VM-attribute sampling all vary between episodes, so a single inference table is one draw from a distribution—across the five seeds, the single-repeat Wilcoxon verdict for QLHH alone ranges from $p = 0.81$ to $p = 10^{-5}$. Second, the five meta-heuristics are Numba-compiled and draw from Numba's internal random generator, which is unaffected by the Python 3.13.14 interpreter-level seeding; we seed it explicitly, making every repeat bit-for-bit reproducible under its documented seed. The per-repeat verdicts are reported alongside the averaged test so that the stability of each conclusion can be inspected directly.

4.2. Paired Comparison: HH-RL Reward Under Trust vs. No-Trust

Figure 1 reports the Nash Total Reward of each HH-RL agent on each of the 20 workflows, with solid bars for the No-Trust suite and hatched bars for the Trust suite.

The figure shows per-workflow shifts in reward that are mixed in direction across agents and workflows. Recomputing the paired deltas as per-workflow means over the five seeded repeats yields the following picture, broken down by agent.

For PDQN, 16 of 20 workflows show a downward mean shift under the trust suite and 4 show an upward shift, with the largest drops on CyberShake_30 ($\Delta = -0.62$), Epigenomics_100 (-0.52), and Inspiral_30 (-0.50), and the largest gains on Sipht_1000 ($+1.28$), Inspiral_1000 ($+1.28$), and CyberShake_1000 ($+0.34$). For DQNHH, 18 of 20 workflows shift

down and 2 shift up; the largest drops are on Inspiral_100 (−0.92), Inspiral_1000 (−0.81), and Sipht_60 (−0.50), and the only gains on CyberShake_1000 (+0.43) and Montage_25 (+0.23). For QLHH the pattern is the mildest: 13 of 20 workflows shift down and 7 shift up, with all shifts small (largest drop −0.34, largest gain +0.44). All deltas are per-workflow means over the five seeded repeats. Across all three agents, every workflow under the trust suite produces a reward that lies in the positive Nash region; no HH-RL agent collapses to the disagreement-shifted reward floor on any workflow.

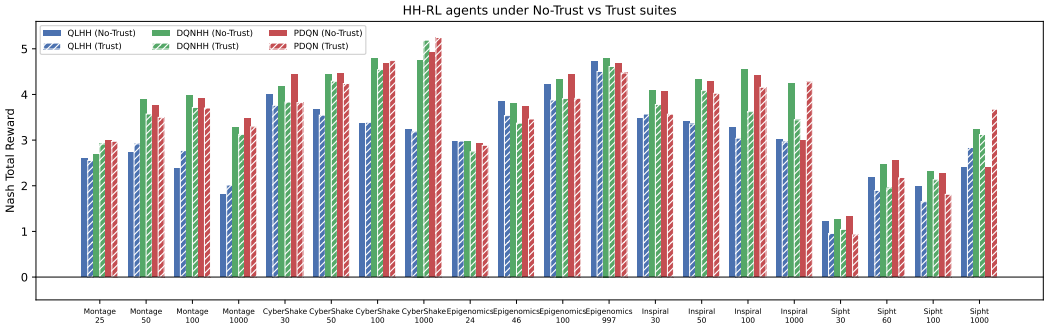

Figure 1. Nash Total Reward of the three HH-RL agents on each of the 20 workflows under the No-Trust (solid) and Trust (hatched) suites. Mean values over five independently seeded repeats (100 inference episodes per cell).

The size-spectrum pattern noted for PDQN is now supported by repeated evidence rather than a single draw: three of its four positive workflows are the 1000-task instance of their family (Sipht_1000, Inspiral_1000, CyberShake_1000), the two largest gains exceed +1.2 Nash units, and these workflows are positive in at least four of the five independent repeats. On the largest action spaces the per-VM trust vector is therefore exploited as a consistently useful contextual cue by the parameterised agent, offsetting—and on those instances reversing—the typical-case attenuation cost. A dedicated multi-instance study at fixed workflow scale remains future work, but the direction of the effect is no longer merely suggestive.

4.3. Wilcoxon Signed-Rank Test and Effect Sizes

To assess whether the trust mechanism produces a statistically detectable change in HH-RL performance, we apply a two-sided Wilcoxon signed-rank test to the 20 per-workflow paired deltas of each agent, where each delta is the mean over the five seeded repeats of $R_w^{\text{Trust}} - R_w^{\text{NoTrust}}$. Figure 2 visualises the per-agent delta distributions; Table 3 reports the averaged test and Table 4 the per-repeat verdicts.

Table 3. Two-sided Wilcoxon signed-rank test on Nash Total Reward across the 20 workflows, comparing Trust and No-Trust suites for each HH-RL agent. Each paired observation is the per-workflow delta averaged over five independently seeded repeats (100 inference episodes per cell). "Up" and "down" count the workflows on which the trust layer raises or lowers the reward. p_{raw} is the unadjusted p-value; $p_{\text{Bonf.}}$ applies a Bonferroni correction for three simultaneous tests; r_{rb} is the rank-biserial correlation effect size. The shift is statistically significant for DQNHH, marginal for PDQN, and not significant for QLHH.

Agent	Median Δ	Mean Δ	Up/Down	Statistic W	p_{raw}	$p_{\text{Bonf.}}$	r_{rb}
PDQN	−0.239	−0.093	4/16	54.0	0.058	0.175	−0.49
DQNHH	−0.240	−0.269	2/18	24.0	0.0014	0.0043	−0.77
QLHH	−0.055	−0.056	7/13	68.0	0.177	0.531	−0.35

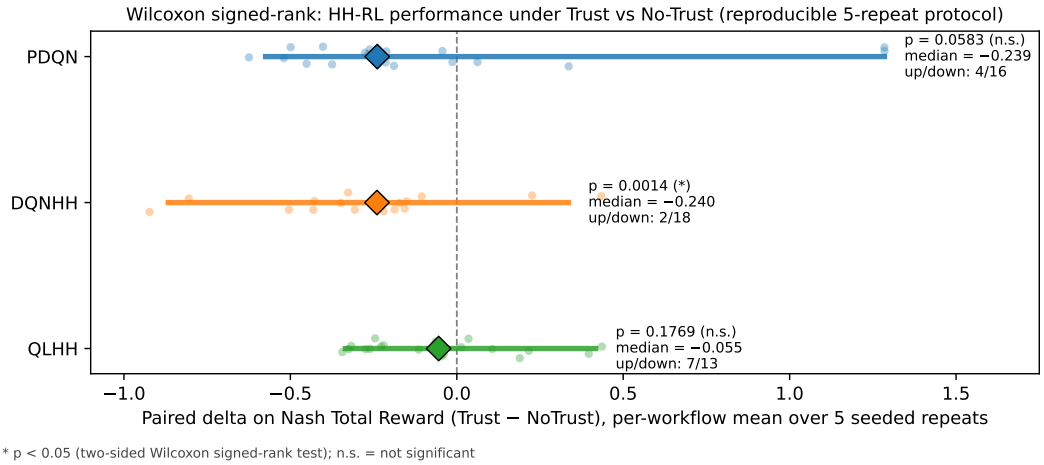

Figure 2. Per-agent distribution of the per-workflow paired deltas $\Delta_w = R_w^{\text{Trust}} - R_w^{\text{NoTrust}}$ across the 20 workflows, each averaged over the five seeded repeats. Markers report the median; bars report the spread; translucent dots show the individual workflow deltas. The asterisk (*) marks a statistically significant shift at $p < 0.05$ (two-sided Wilcoxon signed-rank test); n.s. denotes not significant.

Table 4. Per-repeat single-draw verdicts (median Δ and raw two-sided Wilcoxon p, $n = 20$) for each documented seed. The spread—most visibly QLHH's p ranging from 0.81 to 10^{-5}—demonstrates that a single inference table cannot support a stable significance verdict, motivating the repeat-averaged test of Table 3.

	Seed 42	**Seed 142**	**Seed 242**	**Seed 342**	**Seed 442**
PDQN	$-0.360/0.017$	$-0.136/0.202$	$-0.036/0.409$	$-0.175/0.245$	$-0.334/0.015$
DQNHH	$-0.292/0.0005$	$-0.192/0.064$	$-0.281/0.004$	$-0.220/0.044$	$-0.323/0.0007$
QLHH	$-0.248/0.596$	$+0.020/0.812$	$+0.017/0.498$	$-0.153/0.083$	$-0.219/10^{-5}$

Three observations follow. First, the trust mechanism imposes a small but real absorption cost, and the three agents pay it to different degrees. DQNHH exhibits a statistically significant downward shift (raw $p = 0.0014$, surviving Bonferroni correction at $p = 0.0043$) with a large effect size ($r_{rb} = -0.77$) and a highly consistent direction: 18 of 20 workflows decline, and the per-repeat verdict is significant or near-significant under every seed. PDQN's shift is marginal ($p = 0.058$): its typical-case cost is comparable in median to DQNHH's, but it is offset by large, repeatable gains on the biggest DAGs (Section 4.2), which widen the delta distribution and pull the mean toward zero (-0.093). QLHH shows no significant shift ($p = 0.177$) and the smallest deltas throughout. Second, the magnitude of the cost is bounded and small in context: the median deltas lie in $[-0.24, -0.05]$ against absolute rewards of 2–5 and a between-family separation of 3.3–5.0 Nash units, i.e., an order of magnitude below both. Third, the per-repeat table makes explicit why the repeated protocol is necessary: single-draw verdicts fluctuate across the significance threshold in both directions, and only the averaged test is stable.

The structural interpretation is unchanged in kind but corrected in degree. The two couplings of Section 3.4 that touch the reward—admission filtering, which removes ∼10% of the action space, and the trust-attenuated effective security, which degrades u_{sec} whenever $T_v(t) < 1$—push the reward downward, and that push is now measurable: small everywhere, statistically significant for the fixed-θ neural agent, marginal for the parameterised agent, and absorbed without significant trace by the tabular agent. We therefore no longer describe the extension as cost-free; we describe it as *cheap*—a bounded typical-case cost that does not disturb the family-level scaling separation (Section 4.5)—and, for the parameterised agent on the largest DAGs, a net performance gain rather than a cost.

4.4. Security Utility Under the Trust Layer

Because the trust mechanism re-enters the security utility u_{sec} through the trust-attenuated effective security level of Equation (7), the behaviour of u_{sec} under the trust suite is a direct test of whether the structural Nash-floor safety mechanism interacts benignly with the dynamic trust signal. Figure 3 reports u_{sec} for the three HH-RL agents on each of the 20 workflows.

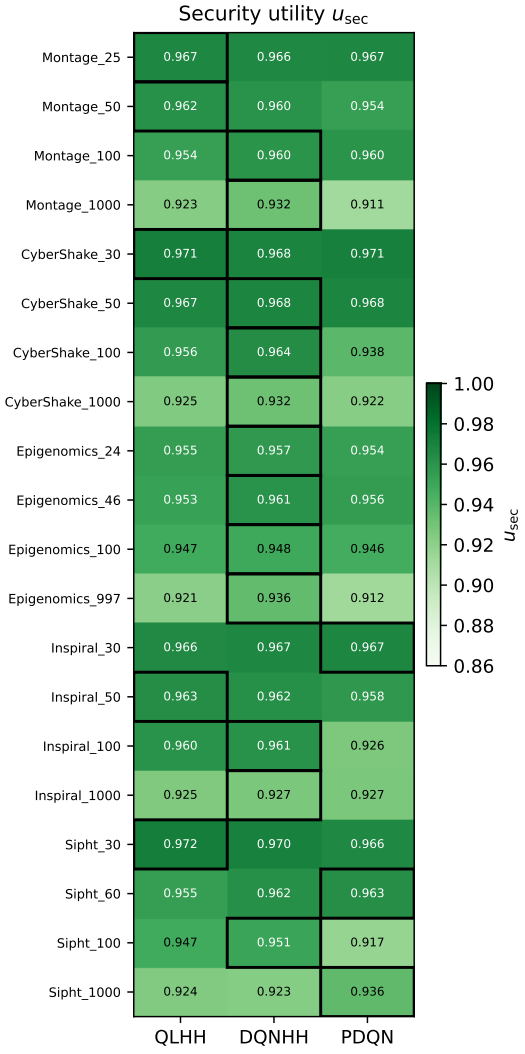

Figure 3. Security utility u_{sec} achieved by each HH-RL agent on each workflow under the Trust suite. Per-row maxima are bordered. Cell values are means over five independently seeded repeats (100 inference episodes per cell).

Computing the per-agent summary across the 20 cells, QLHH attains a median $u_{\text{sec}} = 0.955$ (range 0.921–0.972), DQNHH a median of 0.960 (0.923–0.970), and PDQN a median of 0.954 (0.911–0.971). The worst-case minimum across all 60 (agent, workflow) cells is 0.911, attained by PDQN on Montage_1000. All 60 cells are well above the $d_{\text{sec}} = 0.50$ Nash disagreement floor. All values are means over the five seeded repeats; the seeded sensitivity sweep of Section 4.6 independently confirms the saturation (minimum u_{sec} 0.911, median near 0.956 across all seeds).

Two structural conclusions follow. First, the trust-attenuated coupling $Sec_v^{\text{eff}}(t) = \lfloor Sec_v \cdot T_v(t) \rfloor$ does not, in the regimes tested here, push u_{sec} anywhere near the Nash floor. The base framework's structural floor at $d_{\text{sec}} = 0.50$ retains a comfortable margin under the trust suite, even though the security utility is now mechanically penalised whenever trust drifts below 1. Put differently, $d_{\text{sec}} = 0.50$ operates here as a safety *guardrail* rather

than as an actively binding constraint: it exists to catch a pathological collapse of the security utility, and the trust-attenuated coupling does not, in any tested regime, drive u_{sec} toward it. This is by design, and—importantly—it is not an artifact of the small baseline noise $\sigma = 0.05$. The sensitivity analysis of Section 4.6 shows that even at $\sigma = 0.30$, six times the baseline, the worst-case u_{sec} moves by less than 0.001; the margin to the floor is therefore structural (Section 4.6 identifies the three mechanisms responsible), and a regime in which d_{sec} becomes actively binding would require altering the trust *dynamics* rather than merely raising the noise amplitude. Second, the security utility under the trust suite is strikingly homogeneous across the three HH-RL agents (medians 0.950, 0.954, 0.957). This homogeneity indicates that the high-u_{sec} behaviour is not a property of any one agent's learning dynamics but of the framework's combined defences—zero-trust admission control, the meta-heuristic library's security-aware fitness function, and the Nash floor on u_{sec}—which jointly enforce a strong lower bound on security compliance that survives the addition of the dynamic trust signal.

We should be candid about the flip side of this saturation, which a reader might otherwise raise: because u_{sec} stays near 0.95 almost regardless of scheduling choices, it contributes little *gradient* to the reward and is therefore a weak training signal in the regime studied here. This is consistent with, rather than contrary to, the paper's thesis. The informative training signal comes from the other three utilities (makespan, cost, utilisation), while the trust-attenuated u_{sec} acts primarily as a safety guardrail that penalises only the pathological cases in which trust erosion would push effective security below task requirements. The coupling is thus doing exactly what a security-realism layer should do—constraining without dominating—and making u_{sec} a strong *differentiating* signal would require trust dynamics severe enough to routinely breach task security requirements, which the sensitivity analysis of Section 4.6 shows the present dynamics do not, and which we leave to future work.

4.5. Scaling Behaviour: HH-RL vs. Direct-RL Under Both Suites

A natural concern is whether the trust mechanism degrades the family-level scaling behaviour established under the No-Trust suite, namely that HH-RL delivers a positive Nash reward at every workflow size while the Direct-RL family collapses to the utility floor on the larger DAGs. Figure 4 reports the mean Nash Total Reward broken down by family and workflow size bucket under both suites. The cell counts, between-family gaps, and bucket means reported in this subsection are computed from the repeat-averaged absolute rewards of the seeded protocol (Section 4.1).

Recomputing the bucket means directly from the inference tables, the HH-RL family means under the No-Trust suite are $+3.03$, $+3.60$, $+3.63$, and $+3.59$ across the four size buckets (small, medium-small, medium, large), and $+2.84$, $+3.33$, $+3.42$, $+3.72$ under the Trust suite. The HH-RL family remains uniformly positive across all four size buckets under both suites, with the trust-suite curve sitting only marginally below the no-trust curve and, on the large bucket, marginally above it. The Direct-RL family means are uniformly negative under both suites: -0.26, -0.74, -1.23, -1.25 under No-Trust, and -0.51, -0.69, -1.03, -1.32 under Trust.

Three conclusions follow. First, the trust mechanism does not rescue the Direct-RL family from its collapse at larger sizes: Direct-RL remains uniformly negative. Second, the trust mechanism does not destabilise the HH-RL family's robustness: HH-RL remains uniformly positive. Third, the between-family separation that defines the base framework's central result is preserved unchanged: HH-RL outperforms Direct-RL by between 3.4 and 5.0 Nash units at every size bucket under the Trust suite, comparable to the No-Trust separation. The family-level scaling story survives the trust extension, which is the

strongest single piece of evidence for the mechanism contribution: the trust layer can be composed into the RL-HH machinery without changing what makes the family work.

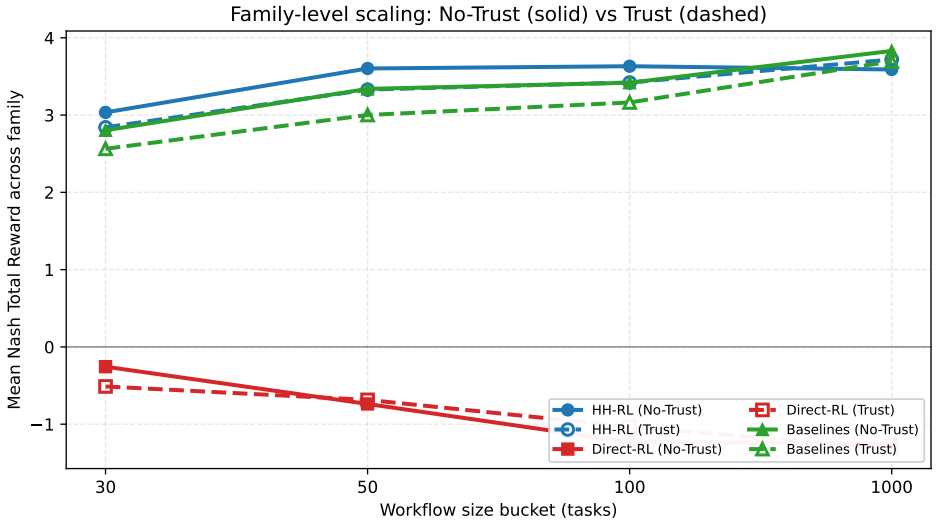

Figure 4. Mean Nash Total Reward as a function of workflow size bucket, separately for the HH-RL family (PDQN, QLHH, DQNHH) and the Direct-RL family (QLEARN, DQN, MultiActionQL, MultiActionDQN), under both the No-Trust and Trust suites. Means are taken across all (workflow type, agent) cells within the family at each size.

The same finding stated at the level of individual (agent, workflow) cells is even more emphatic. Under the trust suite, all 60 HH-RL cells return a positive Nash reward whereas only 15 of the 80 Direct-RL cells do so. Restricting attention to the four 1000-task workflows—where the base framework exhibits the sharpest divergence between the two families—the gap between the best HH-RL agent and the best Direct-RL agent ranges from $+1.16$ Nash units on Montage_1000 to $+6.60$ units on Epigenomics_997, with Direct-RL collapsing to the reward floor of -2.00 on both Epigenomics_997 and Sipht_1000 while the corresponding HH-RL agents return $+4.60$ and $+3.65$ respectively. The HH-RL family's superiority therefore is not a property of the average over workflow sizes but holds uniformly across the size spectrum and grows rather than shrinks at the extremes, exactly the scaling pattern that the hyper-heuristic action-space framing is designed to deliver.

4.6. Sensitivity Analysis: Trust Noise, Zero-Trust Failure Rate, and Multi-Seed Variance

A natural concern is whether the absorption result reported above holds only under the single, manually chosen parameter set ($\sigma = 0.05$, $\rho_{ZT} = 0.10$), or whether it is a stable property of the mechanism across trust regimes. To settle this we conduct a two-level, one-factor-at-a-time sensitivity sweep, re-evaluating the trained hyper-heuristic agents under $\sigma \in \{0.05, 0.20, 0.30\}$ at $\rho_{ZT} = 0.10$, and under $\rho_{ZT} \in \{0.10, 0.30\}$ at $\sigma = 0.05$. Because σ drives the trust random walk of Equation (5) and ρ_{ZT} drives admission failures in $\mathcal{V}^{ZT}$, both perturbations alter the inputs to u_{sec} and the admitted fleet without touching the learned policy, so the sweep isolates the robustness of the result to the trust configuration. To address the concern that single-seed inference underestimates variance, every cell is evaluated over five independent VM-generation seeds and aggregated; the reported dispersion therefore reflects VM-attribute resampling rather than a single draw.

Table 5 and Figure 5 report the outcome, computed under the same seeded protocol as the paired comparison. Raising the trust-walk noise six-fold ($\sigma : 0.05 \rightarrow 0.30$) changes the mean HH Nash reward by at most 0.014 units, with two-sided Wilcoxon $p = 0.18$ and 0.80 against the baseline configuration; tripling the zero-trust failure rate ($\rho_{ZT} : 0.10 \rightarrow 0.30$) changes the mean reward by only $+0.017$ units ($p = 0.21$). All 60 agent–workflow cells

remain in the positive Nash region under every configuration, and the cross-seed reward standard deviation is a modest 0.19–0.28 Nash units. The security utility is likewise stable: its per-configuration minimum moves by less than 0.001 under the largest noise, and its median stays near 0.956. The absorption result is therefore not an artifact of the tuned parameter set—it survives a six-fold change in trust noise and a three-fold change in admission stringency. The slight non-monotonicity across σ (the $\sigma = 0.20$ mean sits marginally below the baseline while $\sigma = 0.30$ sits marginally above) is within cross-seed noise: all $|\Delta| < 0.02$ Nash units and all $p > 0.17$.

Table 5. Sensitivity of the hyper-heuristic family to the trust-walk noise σ and the zero-trust failure rate ρ_{ZT}, aggregated over the three HH-RL agents, the 20 workflows, and five VM seeds (60 agent–workflow cells per configuration). "Pos. cells" counts cells with positive Nash reward; $u_{\text{sec}}^{\min}$ and $u_{\text{sec}}^{\text{med}}$ are the minimum and median security utility across the 60 cells; "SD_{seed}" is the mean cross-seed standard deviation of the reward; and the Wilcoxon p is a two-sided paired test of each configuration's per-cell rewards against the baseline configuration. A six-fold increase in σ and a tripling of ρ_{ZT} leave the mean reward and the security utility essentially unchanged, and all 60 cells remain positive throughout.

Configuration	Mean R	Δ vs. Base	Pos. Cells	$u_{\text{sec}}^{\min}/u_{\text{sec}}^{\text{med}}$	SD_{seed}	Wilcoxon p
$\sigma = 0.05$, $\rho_{ZT} = 0.10$ (base)	3.383	—	60/60	0.911/0.956	0.27	—
$\sigma = 0.20$, $\rho_{ZT} = 0.10$	3.369	−0.014	60/60	0.911/0.956	0.28	0.18
$\sigma = 0.30$, $\rho_{ZT} = 0.10$	3.380	−0.004	60/60	0.911/0.956	0.27	0.80
$\sigma = 0.05$, $\rho_{ZT} = 0.30$	3.400	+0.017	60/60	0.912/0.962	0.19	0.21

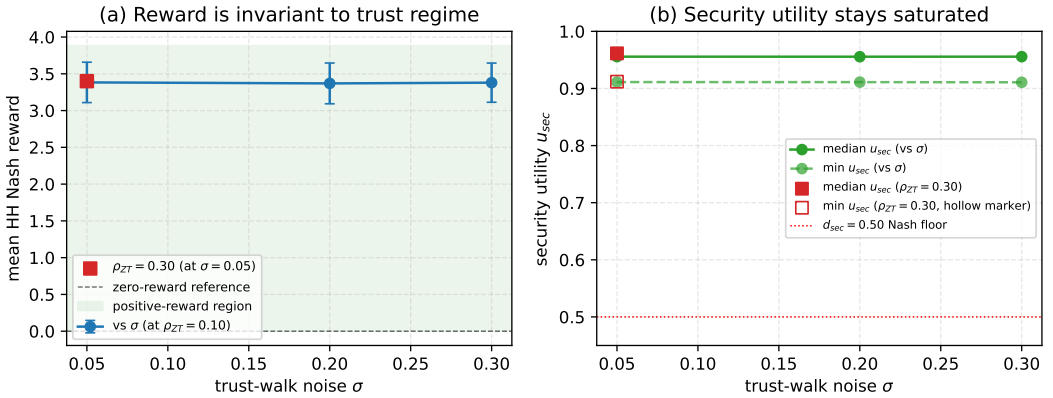

Figure 5. Sensitivity of the hyper-heuristic family to the trust configuration. (**a**) Mean HH Nash reward stays in the positive region and is statistically indistinguishable from the baseline as σ is raised six-fold (error bars: mean cross-seed standard deviation) and as ρ_{ZT} is tripled (square marker). (**b**) The security utility u_{sec} (minimum and median across the 60 cells) remains saturated far above the $d_{\text{sec}} = 0.50$ Nash floor across all configurations.

The near-invariance of u_{sec} deserves comment, because it might otherwise be read as evidence that $\sigma = 0.05$ is simply too small to stress the security coupling. The sweep rules this out directly: even at $\sigma = 0.30$, six times the baseline, u_{sec} barely moves. The stability is structural rather than a consequence of a small σ. Three mechanisms combine to produce it. First, the security penalty in the simulator is normalised against an all-tasks-maximally-violated worst case, so the realised penalty—a small number of high-requirement tasks landing on trust-degraded VMs—is a small fraction of the normaliser. Second, the agents learn to route high-req_sec tasks onto high-Sec_v machines, which structurally minimises violations regardless of trust noise. Third, per-episode trust reset ($T_v(0) = 1$) combined with the slow EMA ($\alpha = 0.1$) keeps the effective security level close to nominal within an episode, so a larger innovation variance does not translate into proportionally larger effective-security degradation. Meaningfully stressing u_{sec} would therefore require changing the trust *dynamics* rather than merely the noise amplitude—for example, persistent

cross-episode trust, a larger adaptation coefficient, or reliability anchoring ($\beta > 0$ in Equation (5))—which we identify as a concrete direction for future work.

4.7. Training Behaviour Under the Trust Suite

To confirm that the inference-time behaviour analysed in the preceding subsections reflects a stable learning process rather than a brittle transient, we examine the per-episode training trajectory of PDQN under the trust suite on a representative workflow size. Figure 6 reports the reward and the four normalised utility components over the $E = 10{,}000$ training episodes on the size-100 bucket of the trust suite.

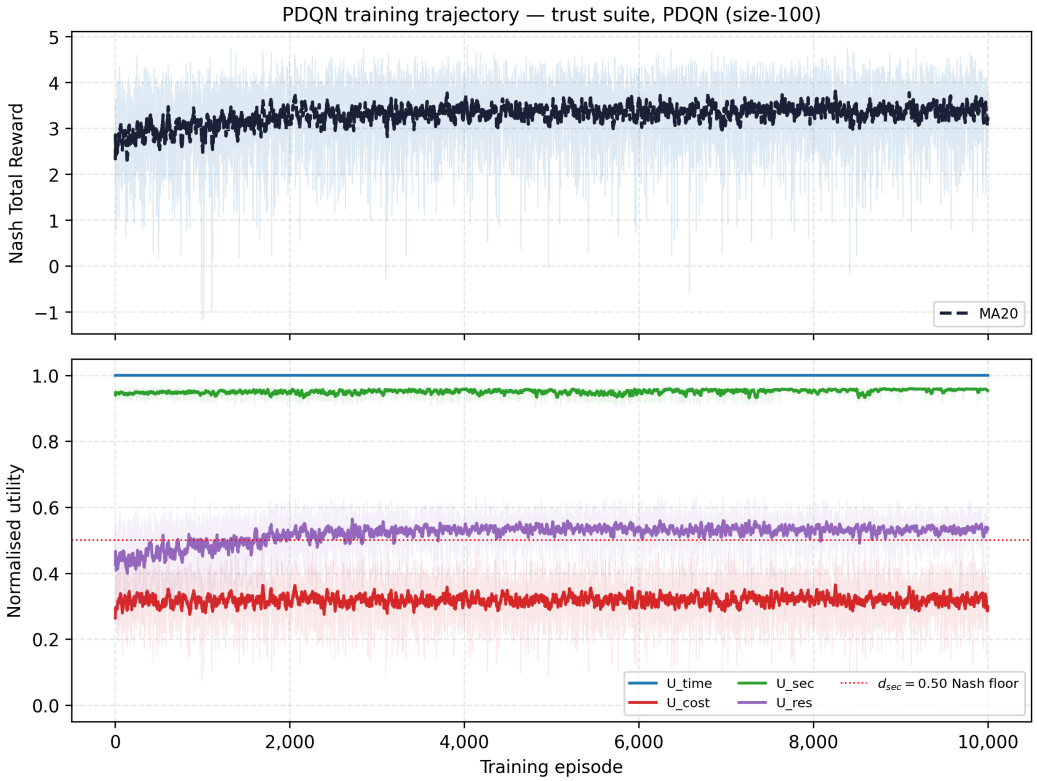

Figure 6. PDQN training trajectory on the size-100 bucket of the trust suite. (**Top**): Per-episode Nash Total Reward (light) and a 20-episode moving average (dashed); the agent enters a stable regime around episode 1500 and the moving average remains in the 3.2–3.6 band thereafter. (**Bottom**): The four normalised utility components. u_{sec} sits around 0.95 rather than at 1.00, reflecting the trust-attenuated effective security coupling; no component approaches its disagreement floor. The qualitative shape of convergence matches that of the no-trust baseline, preserved under the trust layer.

Figure 7 reports the algorithm-selection behaviour over the same training run, matching the analysis presented for the no-trust suite.

Two observations close the empirical analysis. First, the algorithm-selection profile is preserved qualitatively under the trust layer: TABU remains the most frequently chosen optimiser (≈33% of selections), ILS rises to second (≈24%), and the relative ordering of the five meta-heuristics matches the no-trust selection profile. PDQN does not collapse onto a single meta-heuristic when trust is enabled, nor does it abandon any of the five. Second, the θ trajectories show the same heuristic-specific separation seen under no-trust, indicating that the parameter-tuning component of PDQN absorbs the additional trust dimension without breaking the learned per-heuristic regimes. Together, these training-time observations corroborate the inference-time conclusion of Sections 4.3–4.5: the trust mechanism is absorbed by the RL-HH machinery as a contextual perturbation rather than as a structural failure of the learning dynamics.

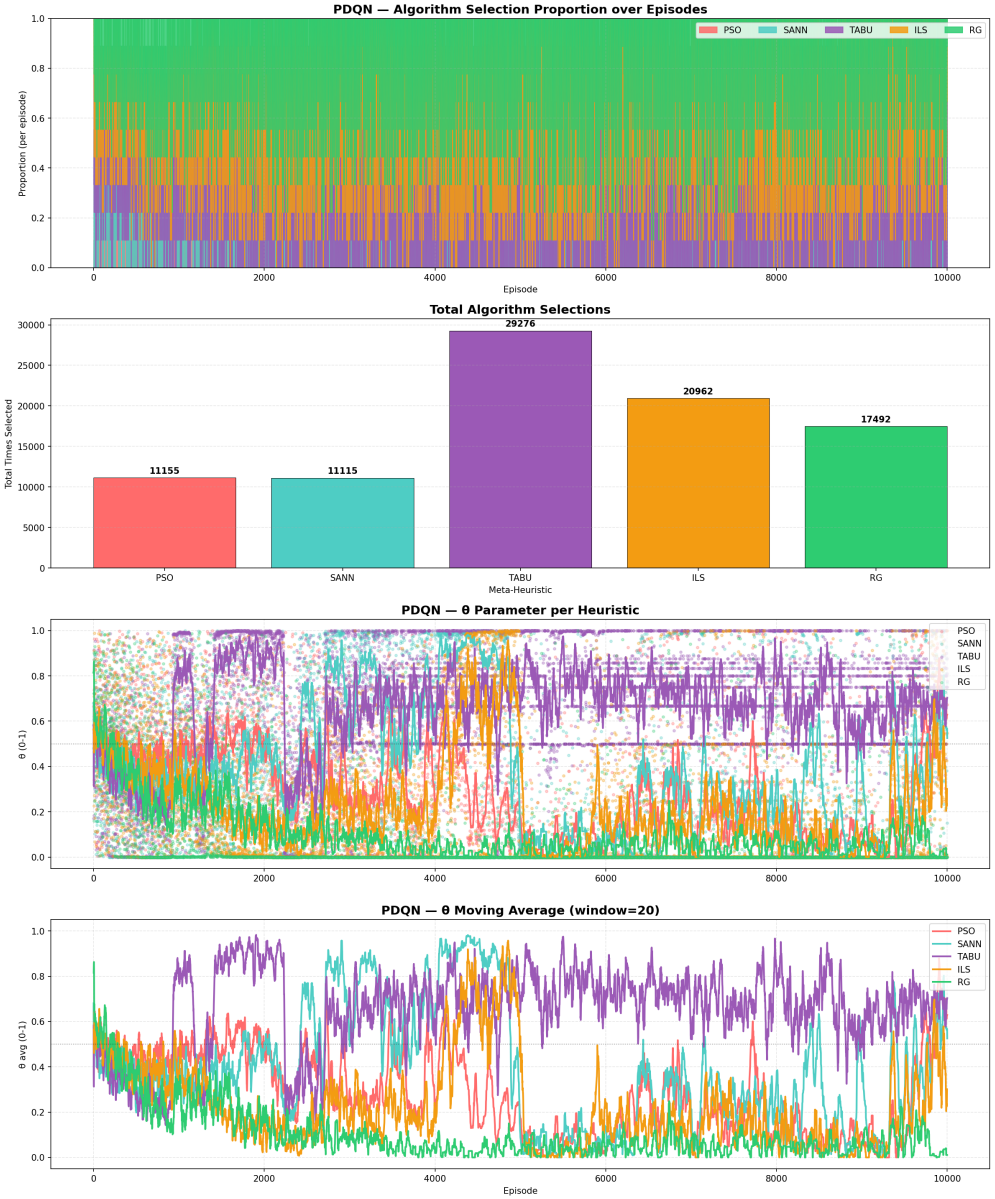

Figure 7. PDQN algorithm-selection and θ-tuning behaviour on the size-100 bucket of the trust suite. (**First**): Per-episode selection proportions of the five meta-heuristics. (**Second**): Total selection counts over the 10,000-episode run (TABU 29,276, ILS 20,962, RG 17,492, PSO 11,155, SANN 11,115). (**Third, Fourth**): Per-episode θ scatter and its 20-episode moving average per heuristic.

These trajectories also speak to a convergence question raised for the state augmentation: does extending the state from $8 + 2K$ to $8 + 3K$ dimensions slow learning or require more episodes to stabilise? The augmentation enlarges only the input layer of the value and policy networks—K additional inputs, i.e., a small increase in first-layer parameters, with no change to network depth, action space, or reward—so *a priori* it should not materially alter convergence. Empirically, Figure 8 places the PDQN reward trajectory under the no-trust baseline (state $8 + 2K$) beside the trust suite (state $8 + 3K$). Both enter their stable regime at approximately the same episode ($\approx$1500); the trust curve settles into a marginally lower band ($\approx$3.1–3.5 versus $\approx$3.3–3.7 Nash units), consistent with the small absorption cost quantified in Section 4.3, and its security-utility component sits slightly lower ($\approx$0.95 versus $\approx$0.97) as expected from the trust attenuation. Crucially, there is no longer transient and no enlarged episode budget under the trust suite: the additional K trust dimensions do not impair convergence.

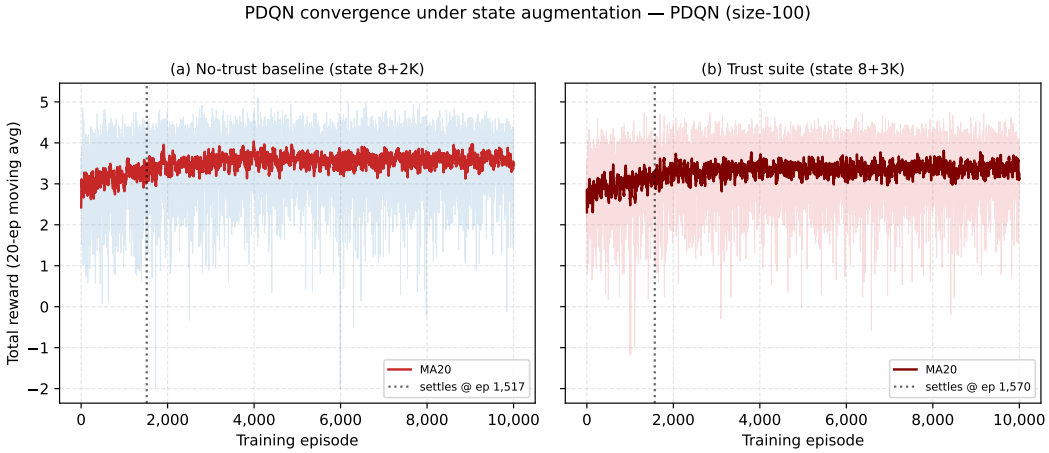

Figure 8. PDQN convergence under state augmentation. Per-episode Nash Total Reward (light) and its 20-episode moving average (MA20, red) over the 10,000-episode training run, for (**a**) the no-trust baseline (state $8 + 2K$) and (**b**) the trust suite (state $8 + 3K$). Both stabilise at approximately the same episode ($\approx$1500); the trust run settles into a marginally lower reward band, consistent with the small absorption cost, with no slower convergence and no enlarged episode budget from the additional K trust dimensions.

5. Conclusions and Future Work

This paper has presented the mechanism design required to integrate two security-realism layers into the base RL hyper-heuristic scheduling framework. The mechanism consists of four components composed in a drop-in fashion on top of the base framework: a five-stage zero-trust verification pipeline that filters candidate VMs through sequential admission checks; a bounded, non-stationary dynamic trust signal modelled as an exponentially weighted moving average of an observation that can be anchored to VM execution reliability; a state-vector augmentation that exposes the per-VM trust vector to the RL agent at every decision epoch; and a trust-attenuated effective security coupling that propagates the dynamic trust value into the security utility used by the Nash reward. The Nash Social Welfare reward functional—its weights, disagreement points, and shift—is inherited from the base framework without modification, so any change in observed behaviour can be traced to the four explicit coupling points rather than to hidden reward redesign.

The empirical evaluation on 20 Pegasus workflow instances supports three claims. First and most importantly, the hyper-heuristic RL family's superiority over the direct task-to-VM family—established under the trust-free baseline—survives the trust extension intact, and indeed becomes more emphatic when stated at the cell level: HH-RL agents deliver a positive Nash reward on 60 of 60 (agent, workflow) cells under the trust suite, while Direct-RL agents do so on only 15 of 80 cells, with the between-family gap on the four 1000-task workflows ranging from $+1.16$ to $+6.60$ Nash units. The family-level scaling story of the base framework—HH-RL uniformly positive at every workflow size, Direct-RL uniformly collapsed—is therefore preserved under the trust extension, and preserved with the same magnitude of separation as in the no-trust baseline. Second, under a reproducible five-repeat evaluation protocol the trust mechanism imposes a small absorption cost that differs by agent: statistically significant for DQNHH (two-sided Wilcoxon $p = 0.0014$, $r_{rb} = -0.77$; median paired delta -0.24), marginal for PDQN ($p = 0.058$; median -0.24, offset by repeatable gains exceeding $+1.2$ Nash units on the largest DAGs), and not significant for QLHH ($p = 0.18$; median -0.05). The median costs are an order of magnitude smaller than the absolute reward levels and the between-family separation. Third, the security utility under the trust-attenuated coupling remains above 0.91 on every (agent, workflow) cell, with a median of approximately 0.95, indicating that the structural Nash floor at

$d_{\text{sec}} = 0.50$ retains a comfortable margin under the dynamic trust signal. Together these findings support the mechanism claim: the RL-HH machinery absorbs the trust extension without sacrificing its multi-objective balance or, crucially, the family-level superiority that motivates choosing a hyper-heuristic architecture in the first place.

We stress what this result does and does not establish. Both security-realism layers introduce forces that must, on any honest accounting, cost something: admission control removes roughly a tenth of the action space, and the trust-attenuated coupling mechanically lowers u_{sec} whenever trust drifts below one. The corrected, reproducible measurement shows that this cost is real but bounded—a median of 0.05 to 0.24 Nash units against rewards of 2–5—that one of the three agents pays it at statistical significance while another converts the trust signal into net gains on the largest instances, and that the family-level scaling separation that motivates the hyper-heuristic architecture is preserved throughout. In security engineering the relevant question is rarely whether a realism layer is free but whether its cost is small, predictable, and structurally contained; the contribution of this paper is establishing, under a seeded and repeatable protocol, that for the zero-trust/dynamic-trust pair composed into the RL-HH loop, it is all three.

Several limitations of the present study point naturally to future work.

(1) Exogenous trust dynamics. The observed trust estimate $\widehat{T}_v(t)$ used in the main experiments reduces to a clipped Gaussian random walk ($\beta = 0$ in Equation (5)); the framework admits anchoring to the VM execution reliability R_v through $\beta > 0$, but the empirical sweep over β is deferred. Coupling $\widehat{T}_v$ to a richer anomaly-detection backend (integrity attestation, side-channel observations, or learned anomaly scores) would test whether the agent can learn behaviour that exploits structured rather than purely stochastic trust dynamics.

(2) Fixed zero-trust failure rate. The main experiments fix the per-VM verification failure probability at $\rho_{ZT} = 0.10$. The sensitivity analysis of Section 4.6 now sweeps this rate to $\rho_{ZT} = 0.30$ and finds the reported results stable. Pushing ρ_{ZT} high enough to sparsify the verified fleet below the parallelism implied by the workflow's level cardinality—where a phase transition is expected—remains future work.

(3) Single observation regime. The trust signal is exposed to the agent with no delay and no observation noise—the agent reads $T_v(t)$ directly. A more realistic deployment would expose only delayed and partially observable trust estimates, which would call for partially observable MDP techniques beyond the standard PDQN architecture used here.

(4) Stylised verification stages. Each γ_m in the zero-trust pipeline is implemented as a binary flag rather than as a substantive runtime check. Substituting individual stages with richer mechanisms (cryptographic attestation, telemetry-driven policy compliance, blockchain-backed identity registration) is a deployment-time choice that the framework is intended to accommodate without further structural changes.

(5) Single random seed at fleet instantiation. VM properties (MIPS, security level, reliability, and the ZT failure draw) are seeded once per workflow for the headline paired statistics of Section 4.3. The sensitivity analysis of Section 4.6 now aggregates over five fleet seeds; a full multi-seed re-evaluation of the paired Wilcoxon tables across the entire suite would further refine them.

Outlook.

The mechanism contribution of this paper is composable in a strong sense: any RL agent that operates on the base framework's level state s_t can be evaluated on the trust suite by feeding it $s'_t = [s_t, \mathbf{T}_t]$ and the trust-attenuated u_{sec}, with no further changes to the learning loop or to the Nash reward functional. We expect the same mechanism to slot unchanged into adjacent settings—mobile-edge workflow scheduling, multi-cloud federated scheduling, and IoT-cloud hybrid pipelines—where the same combination of

admission-control feasibility and time-varying resource trustworthiness arises. Validating the mechanism in those settings, and replacing the stylised observation model with a telemetry-driven one, are the natural next steps.

Author Contributions: Conceptualization, H.A.S., S.T.F.A.-J. and E.T.Y.; methodology, H.A.S.; software, H.A.S.; validation, H.A.S., E.T.Y. and O.A.A.; formal analysis, H.A.S.; investigation, H.A.S.; writing—original draft preparation, H.A.S.; writing—review and editing, S.T.F.A.-J., E.T.Y. and O.A.A.; supervision, S.T.F.A.-J. and E.T.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the British Council through the Going Global Partnerships—Researcher Challenges Grant (Grant Reference: RCG2024-003) as part of the project "Supporting Women's Resilience to Climate Change through Technology and Education", led by Liverpool John Moores University in partnership with Al-Maarif University.

Data Availability Statement: The Pegasus workflow instances used in this study are publicly available; the implementation and generated result files are available from the corresponding author upon reasonable request.

Acknowledgments: This research was supported by the British Council through the Going Global Partnerships—Researcher Challenges Grant (Grant Reference: RCG2024-003) as part of the project "Supporting Women's Resilience to Climate Change through Technology and Education", led by Liverpool John Moores University in partnership with Al-Maarif University.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Comparative Analysis of Security-Aware Workflow Scheduling

Table A1 positions this paper against representative prior work in security-aware workflow scheduling. The columns are the comparison dimensions of Table 1. Two columns are new relative to typical survey tables: ZT marks works that include an explicit zero-trust admission layer, and DynT marks works that model trust as a time-varying signal inside the scheduling loop.

Table A1. Comparative analysis of security-aware scientific workflow scheduling approaches. ✓ = supported, – = not addressed. ZT = zero-trust admission; DynT = dynamic/time-varying trust signal modelled inside the scheduler.

Work	Year	T&C	En	Sec	B/D	RL/HH	MH	Sel-MH	Nash	ZT	DynT	Remarks
Sujana et al. [8]	2020	✓	–	✓	–	–	–	–	–	–	–	Fuzzy VM selection
Udatha & Lakshmeeswari [17]	2021	✓	–	–	–	–	✓	–	–	–	–	PSO baseline
Dorsala et al. [22]	2021	–	–	✓	–	–	–	–	–	–	✓	Blockchain trust
El-Kassabi et al. [14]	2023	–	–	✓	–	✓	–	–	–	–	–	Runtime enforcement
Reddy & Kumar [9]	2023	✓	–	✓	–	–	✓	–	–	–	–	Hybrid heuristic
Banerjee & Tekawade [7]	2023	✓	–	✓	✓	–	–	–	–	–	–	Multi-cloud
Zade et al. [11]	2023	✓	–	✓	–	–	✓	–	–	–	–	Adversary model
Udomkasemsub et al. [16]	2023	✓	–	–	✓	✓	✓	✓	–	–	–	RL hyper-heuristic
Roy et al. [12]	2024	✓	–	✓	–	–	✓	–	–	–	–	Multi-workflow
Li et al. [1]	2024	✓	–	✓	–	–	–	–	–	–	–	Cost–security trade-off
Krishna & Mangalampalli [15]	2024	✓	–	–	✓	✓	–	–	–	–	–	A3C RL
John [13]	2025	✓	✓	✓	–	✓	–	–	–	–	–	Energy–security
Abdi et al. [6]	2025	✓	–	✓	✓	–	–	–	–	–	–	MILP model
Tharani et al. [10]	2025	✓	✓	✓	–	–	–	–	–	–	–	Fuzzy system
He et al. [23]	2025	✓	–	✓	✓	✓	–	–	–	–	–	Deep RL
Bajaher et al. [5]	2025	✓	–	–	✓	✓	✓	✓	–	–	–	RL-HH
Wang et al. [4]	2025	–	–	✓	–	✓	–	–	–	✓	✓	Access control only
Hammouti et al. [24]	2025	✓	–	✓	✓	–	✓	–	–	–	–	Rescheduling
This work	2026	✓	–	✓	–	✓	✓	✓	✓	✓	✓	Zero-trust + DynT + Nash

References

1. Li, L.; Zhou, C.; Cong, P.; Shen, Y.; Zhou, J.; Wei, T. Makespan and Security-Aware Workflow Scheduling for Cloud Service Cost Minimization. *IEEE Trans. Cloud Comput.* **2024**, *12*, 609–624. [CrossRef]
2. Saeed, H.A.; Al-Janabi, S.T.F.; Yassen, E.T.; Aldhaibani, O.A. Survey on Secure Scientific Workflow Scheduling in Cloud Environments. *Future Internet* **2025**, *17*, 51. [CrossRef]
3. Soveizi, N. Security and Privacy Concerns in Cloud-based Scientific and Business Workflows. Ph.D. Thesis, University of Groningen, Groningen, The Netherlands, 2025. [CrossRef]
4. Wang, R.; Li, C.; Zhang, K.; Tu, B. Zero-Trust Based Dynamic Access Control for Cloud Computing. *Cybersecurity* **2025**, *8*, 12. [CrossRef]
5. Bajaher, A.S.; Abdul Hamid, N.A.W.; Ahmad, I.; Hanapi, Z.M. Predictive State-Aware Deep Reinforcement Learning with Hyper-Heuristic for Resolving Conflicting Objectives in Scientific Workflow Scheduling. *IEEE Access* **2025**, *13*, 176460–176481. [CrossRef]
6. Abdi, S.; Ashjaei, M.; Mubeen, S. Deadline-constrained security-aware workflow scheduling in hybrid cloud architecture. *Future Gener. Comput. Syst.* **2025**, *162*, 107466. [CrossRef]
7. Tekawade, A.; Banerjee, S. Cost and Reliability Aware Scheduling of Workflows Across Multiple Clouds with Security Constraints. *arXiv* **2023**, arXiv:2304.00313. [CrossRef]
8. Sujana, J.A.J.; Revathi, T.; Joshua Rajanayagam, S. Fuzzy-based Security-Driven Optimistic Scheduling of Scientific Workflows in Cloud Computing. *IETE J. Res.* **2020**, *66*, 224–241. [CrossRef]
9. Narendrababu Reddy, G.; Phani Kumar, S. Multi-objective secure aware workflow scheduling algorithm in cloud computing based on hybrid optimization algorithm. *Web Intell.* **2023**, *21*, 385–405. [CrossRef]
10. Tharani, P.; Manimala, K.; Kalpana, A.M. Fuzzy Scheduling of Scientific Workflows with Energy and Security Constraints in Cloud. *ICTACT J. Soft Comput.* **2025**, *16*, 3914–3921. [CrossRef]
11. Zade, B.M.H.; Javidi, M.M.; Mansouri, N. An Improved Caledonian Crow Learning Algorithm Based on Ring Topology for Security-Aware Workflow Scheduling in Cloud Computing. *Peer-to-Peer Netw. Appl.* **2023**, *16*, 2929–2984. [CrossRef]
12. Roy, S.; Gharote, M.; Ramamurthy, A.; Pawar, A.; Lodha, S. Security-Aware Scheduling of Multiple Scientific Workflows in Cloud. In *Proceedings of the Service-Oriented Computing—ICSOC 2024, Tunis, Tunisia, 3–6 December 2024*; Springer: Cham, Switzerland, 2024. [CrossRef]
13. John, B. Security-Aware and Energy-Efficient Task Scheduling for Mobile Cloud Computing. *ResearchGate Prepr.* **2025**. [CrossRef]
14. El-Kassabi, H.T.; Serhani, M.A.; Masud, M.M.; Shuaib, K.; Khalil, K. Deep learning approach to security enforcement in cloud workflow orchestration. *J. Cloud Comput.* **2023**, *12*, 10. [CrossRef] [PubMed]
15. Krishna, M.S.R.; Mangalampalli, S.S. PWSA3C: Prioritized Workflow Scheduler in Cloud Computing Using Asynchronous Advantage Actor Critic (A3C) Algorithm. *IEEE Access* **2024**, *12*, 127976–127992. [CrossRef]
16. Udomkasemsub, O.; Sirinaovakul, B.; Achalakul, T. PHH: Policy-Based Hyper-Heuristic with Reinforcement Learning. *IEEE Access* **2023**, *11*, 52026–52049. [CrossRef]
17. Udatha, C.; Lakshmeeswari, G. Multi-objective based Cloud Task Scheduling Model with Improved Particle Swarm Optimization. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 243–248. [CrossRef]
18. Chen, Z.; Xiong, B.; Chen, X.; Min, G.; Li, J. Joint Computation Offloading and Resource Allocation in Multi-Edge Smart Communities with Personalized Federated Deep Reinforcement Learning. *IEEE Trans. Mob. Comput.* **2024**, *23*, 11604–11619. [CrossRef]
19. Chen, Z.; Liang, J.; Yu, Z.; Cheng, H.; Min, G.; Li, J. Resilient Collaborative Caching for Multi-Edge Systems With Robust Federated Deep Learning. *IEEE Trans. Netw.* **2025**, *33*, 654–669. [CrossRef]
20. Chen, Z.; Yu, Z. Intelligent Offloading in Blockchain-Based Mobile Crowdsensing Using Deep Reinforcement Learning. *IEEE Commun. Mag.* **2023**, *61*, 118–123. [CrossRef]
21. Chen, Z.; Zheng, J.; Cheng, H.; Min, G.; Ning, Z.; Li, J.; Zhang, Y. Joint Service Caching and Resource Allocation in DT-Empowered Cloud-Edge Networks Via MARL with Hierarchical Knowledge Transfer. *IEEE Trans. Mob. Comput.* 2026, *early access*. [CrossRef]
22. Dorsala, M.R.; Sastry, V.N.; Chapram, S. Blockchain-based Solutions for Cloud Computing: A Survey. *J. Netw. Comput. Appl.* **2021**, *196*, 103246. [CrossRef]
23. He, H.; Gu, Y.; Hu, Y.; Fang, F.; Ning, X.; Chen, X.; Cheng, L. Real-time workflow scheduling in hybrid clouds with privacy and security constraints: A deep reinforcement learning approach. *Expert Syst. Appl.* **2025**, *278*, 127376. [CrossRef]
24. Hammouti, S.; Yagoubi, B.; Makhlouf, S.A. HyEcoSec: Hybrid Cloud Economic and Secure Workflow Scheduling System. *ECTI Trans. Comput. Inf. Technol.* **2025**, *19*, 485–500. [CrossRef]