Self-Healing Service Operations: An AI-Driven Causal Process-Mining Control Layer for SLA-Optimized Ticket Workflows

Arunraju Chinnaraju

Doctorate in Business Administration, Westcliff University, USA

DOI: https://doi.org/10.47772/IJRISS.2026.10190018

Received: 16 January 2026; Accepted: 21 January 2026; Published: 14 February 2026

ABSTRACT

Enterprise service operations are orchestrated through ticket/workflow systems that manage incidents & support requests under strict SLA constraints. Current automation approaches remain predominantly predictive static using classification models/routing heuristics that cannot adapt system-wide operational policies due to non-stationarity of demand and resource contention. This paper suggests a self-healing framework for enterprise service operations built around an ai-driven closed loop operational control layer (combining process mining, constrained multi-agent reinforcement learning, causal process analytics). The control layer of the service workflow is modelled as an observable Markov decision process (MDP), however it is partially observable due to the use of centralized-training decentralized-execution multi agent reinforcement learning (MARL) that decomposes operational control across queues, skills, and workflow stage. Constrained policy gradient-based MARL is used for the policy learning component; this includes Lagrangian SLA risk constraint coordination for MAPPO style coordination; thereby allowing for adaptive self-healing policies, as well as action policies (skill-based assignment, dynamic priority assignment, escalation gate assignment, batch assignment, automation trigger assignment, and intake throttle assignment). The process mining methodology is utilized to generate executable workflow models, create state abstraction, and develop non-stationary transition dynamics from event logs, which are the result of observed operational behavior as opposed to assumed process models.

To ensure accountability and managerial trust, the architecture incorporates additional "causal process analysis" layer in this design that will allow users to perform both a counter-factual process replay and quasiexperimental impact estimations (including interrupted time series and difference-in-differences design) to assess how much a specific intervention impacts the breach of Service Level Agreement (SLA), time to resolve, and aging of backlogs and reopens. The framework has a "governance enforcement" mechanism (policy constrained learning architecture) which includes probabilistic SLA bound checking, audit log tracking, human-on-the-loop override mechanisms and deployment safeguarding mechanisms. Performance will be evaluated from Open Incident Management and Process Mining data sets using three different metrics; breach probability of SLA, average Time to Resolve (TTR) and backlog stability and robustness of the backlogs during high volume demand shock events. This proposed framework will demonstrate how governance-aware AI control can convert static ticket workflow into dynamic, auditable, self-healing service operations.

Keywords: Self-Healing Service Operations; Multi-Agent Reinforcement Learning; MAPPO; Constrained Reinforcement Learning; Process Mining; Causal Process Analytics; SLA Optimization; AIOps; Operational Control Systems

INTRODUCTION

Large-scale enterprise ticket and workflow systems manage enterprise critical processes including incident management, customer service, human resources, procurement, and compliance (Gans et al., 2003; Sampson & Froehle, 2006). These systems represent the main interface through which work enters into an organization and is assigned, prioritized, routed, escalated and resolved (Gans et al., 2003; Sampson & Froehle, 2006). Although many organizations rely heavily on these systems for their business success, most ticketing systems use only

simple static automation mechanisms, such as rule-based routing and queuing thresholds or predictive triage models (Lingzhe Zhang et al., 2025). However, when demand for services fluctuates over time, both these types of automation are unable to prevent system wide problems caused by an increase in operational load (Jennings et al., 1996).

One of the defining characteristics of Service Operations is the existence of failure cascades that are created by the interdependencies among workflow elements (van der Aalst, 2016). Local congestion at one point in a workflow will create a ripple effect downstream, the use of escalation policies will cause an increase in load rather than an alleviation of the load, and rework loops caused by a premature closure of a case or by misassigning a task will create additional load that contributes to an increasing backlog (Suriadi et al., 2017). All of these issues result in non-linear SLA breach cascades, resolution instability and longer times to resolve issues during periods of increased demand or during major outages. It is important to note that the failures mentioned above are not solely due to poor classification of tickets, but the lack of a global control mechanism that can make decisions regarding how to assign cases to available personnel with the appropriate skill set at various points in the workflow under conditions of uncertainty (Gans et al., 2003).

Current AI-based approaches to ticket handling view the process of ticket handling as primarily a predictive inferential problem, where the goal is to predict the best possible category classification, assignment, or expected resolution time (Xu et al., 2018). While the models used to make predictions about categories and assignments may be better than previous models, they do not provide a complete closed-loop control solution. Predictive models are made independently of the impact of their decisions downstream, and the model's parameters are either fixed statically or adjusted manually (Lingzhe Zhang et al., 2025). Therefore, predictive triage systems have no ability to adjust the operational policy of the system in response to feedback signals such as developing bottlenecks, escalating SLA risk, or changes in the internal workload of the system. Thus, there is a significant disconnect between prediction and control inherent within all current automation paradigms (Castillo et al., 2021).

Therefore, the principal gap in current Service Operation Platforms is not the lack of predictive intelligence, but rather the lack of a system-wide control abstraction (van der Aalst, 2016). The current generation of platforms lacks a closed-loop operational control layer that learns the actual workflow dynamics from execution data, adjusts the operational policy of the platform under explicit SLA risk constraints, and allows for the causal attribution of the impact of its operational policy adjustments (Lingzhe Zhang et al., 2025). Therefore, without this type of control layer, the automation provided by the platform is reactive, brittle under non-stationary demand patterns, and opaque from a governance perspective (Lingzhe Zhang et al., 2025).

This paper formally defines service operation as a closed-loop operational control problem, where the objective is to control the behavior of the entire workflow process end-to-end rather than to optimize each decision individually (Gans et al., 2003). The operational scope includes IT service management, enterprise support functions, and administrative workflows characterized by high volume, heterogeneous tasks, and strict SLA requirements (van der Aalst, 2016). Self-healing is defined as the continuous adjustment of operational policies based upon the observed results of prior executions, thereby allowing the system to automatically alleviate any degradation, restore stability to performance, and recover from an overloaded condition without operator intervention (Belanche et al., 2020). Self-healing is achieved via feedback-driven policy updates that adjust the assignment, prioritization, escalation, batching, and intake controls of the system in response to changing conditions of the system (Chi et al., 2021).

By viewing ticket and workflow systems as dynamic, partially observable control systems, this research identifies the gap between current service operation platforms and proposes a unified control architecture that combines process mining, constrained multi-agent reinforcement learning, and causal policy evaluation (van der Aalst, 2016). The proposed architecture allows the continuous adaptation of operational policies under explicit SLA risk constraints while allowing for auditability and governance through causal accountability (van der Aalst, 2016). By providing an architecture that supports adaptive, policy-driven operational control and closes the gap between static, prediction-centric automation and dynamic, policy-driven operational control,

the research described in this paper moves service operations forward and creates a foundation for AI-driven self-healing service systems (Li et al., 2021; Gursoy et al., 2019; Lu et al., 2019).

Formalization: Sla-Constrained Operational Control

Enterprise service operations mediated by ticket and workflow systems exhibit complex dynamics driven by stochastic arrivals, heterogeneous task characteristics, shared human and automated resources, and tightly coupled escalation policies (ross, 1989; howard & matheson, 1972). Decisions taken at individual workflow stages influence downstream congestion, rework probability, and compliance with service level agreement requirements. Consequently, these properties necessitate a formal control-theoretic representation that captures uncertainty, delays in outcomes, and risk sensitivity, while remaining compatible with real-world enterprise execution environments (altman, 1999; shen & zhang, 2013). The following section provides the formal framework supporting the proposed architecture by defining service operations as a constrained, partially observable control problem and specifying the optimization objective, constraints, and decomposition principles guiding subsequent architectural and algorithmic design.

Service Operations as a Constrained Partially Observable Control Problem: The operational environment is modeled as a constrained partially observable markov decision process, reflecting the fact that the controller does not directly observe the full latent system state at the time of decision making (monahan, 1982; kaelbling et al., 1998). Observable signals include ticket metadata, queue lengths, agent skill availability, priority distributions, and partial historical execution summaries. Latent operational factors include evolving agent fatigue, hidden backlog pressure, and inter-stage coupling effects which are only indirectly revealed through downstream performance outcomes. Actions correspond to operational policy interventions rather than isolated transactional decisions. These interventions include assignment and reassignment across teams, dynamic priority adjustment, escalation gating, batching and deferral strategies, automation triggering, and temporary intake throttling. Importantly, these actions modify the operational policy surface itself, shaping future state evolution rather than producing only immediate effects. The reward function is defined at the system level, aggregating long-horizon measures of resolution efficiency, backlog stability, and rework minimization. Service quality is not embedded directly as a reward term. Instead, SLA compliance is treated as an explicit constraint, reflecting the regulated and risk-sensitive nature of enterprise service operations. This separation ensures that optimization incentives do not implicitly trade service reliability for short-term efficiency gains (altman, 1999).

Non-Stationarity and Delayed Outcome Structure: Service operations are inherently non-stationary. Ticket arrival processes vary with seasonality, business cycles, and external disruptions. Workforce capacity fluctuates due to scheduling, attrition, and skill availability. Escalation and automation policies introduce feedback effects that alter future system dynamics. Therefore, transition probabilities cannot be assumed fixed and must be continuously infused from execution data (zhu et al., 2023). Additionally, key performance indicators such as SLA compliance, resolution lead time, and reopen rates are delayed outcomes, observable only after workflow completion. This introduces a temporal credit assignment problem, where the impact of a policy intervention may not be observable until several decision epochs later. Effective operational control therefore requires policies that anticipate downstream effects and optimize long-horizon performance rather than immediate metrics. Consequently, control decisions must operate over belief states that summarize latent operational conditions and evolving transition dynamics, which are continuously updated based on observed execution traces rather than assumed to be directly observable (kaelbling et al., 1998).

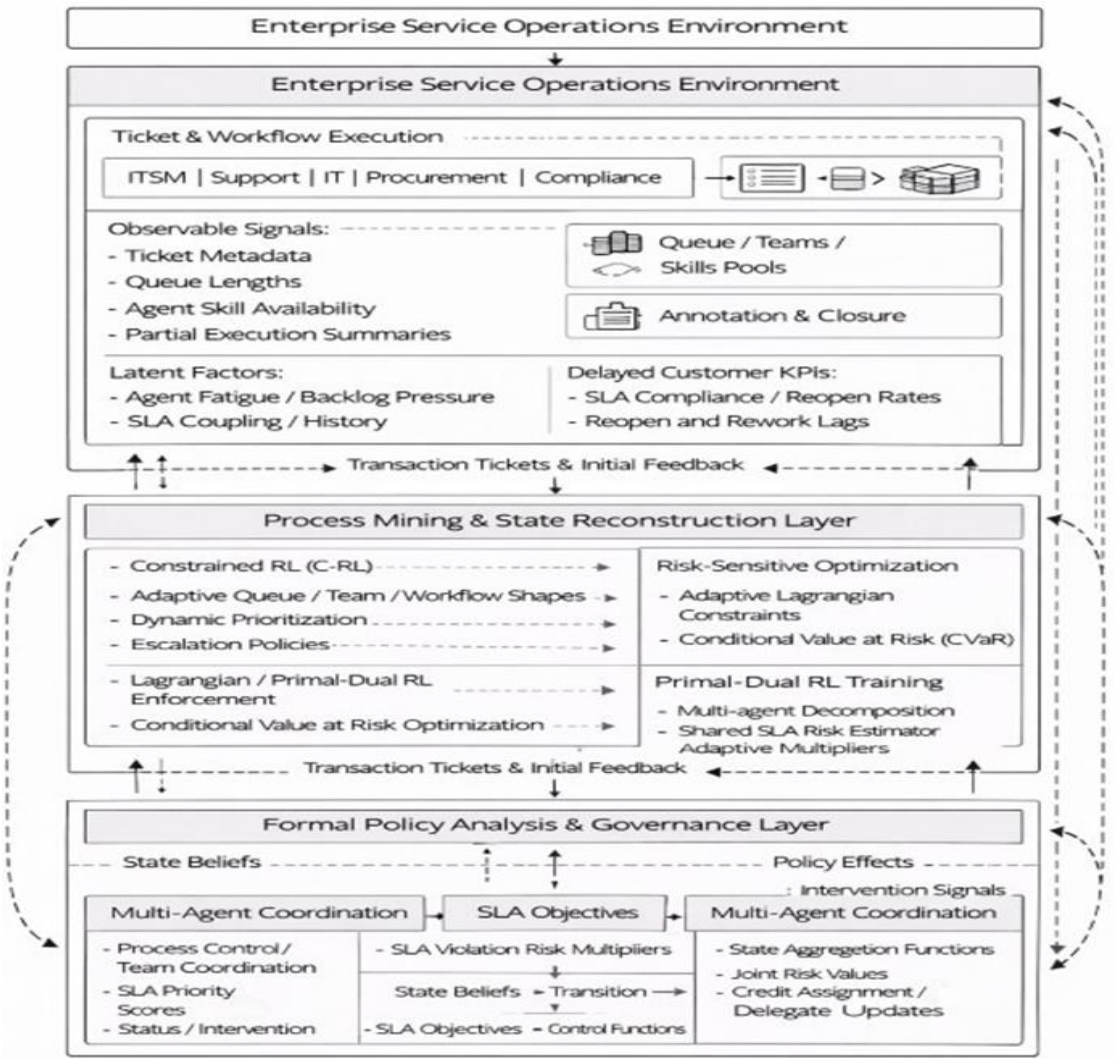
SLA-Constrained Reinforcement Learning Formulation: Policy optimization is performed using constrained reinforcement learning, where SLA compliance is enforced through explicit constraint handling mechanisms (achiam et al., 2017; nakao & fujisaki, 2020). The primary formulation uses a lagrangian or primal-dual approach, whereby SLA risk constraints are incorporated into the optimization objective via adaptive multipliers. These multipliers increase when observed SLA violations exceed acceptable bounds and decrease when the system operates within safe margins, allowing automatic risk-aware policy adaptation under uncertainty (chow et al., 2018). To address rare but severe service failures, the framework supports risksensitive optimization objectives, including conditional value at risk constraints (howard & matheson,

1972). This allows the controller to limit exposure to extreme backlog growth or cascading SLA breaches during high-load scenarios. As a result, operational policies are optimized not only for average-case efficiency but also for robustness under tail-risk conditions, ensuring stability and reliability in volatile enterprise environments (shen & zhang, 2013).

Multi-Agent Decomposition and Coordination: Due to the scale and heterogeneity of enterprise workflows, the control problem is decomposed into a multi-agent formulation, where individual agents correspond to queues, functional teams, or workflow stages (zhang et al., 2021). Each agent operates on local observations and selects actions within its scope, while coordination mechanisms ensure alignment with global SLA and system-level performance objectives. Training follows a centralized training, decentralized execution paradigm, enabling joint optimization of interacting policies while maintaining scalable execution under partial observability. Shared constraints and coordination signals ensure that local actions collectively satisfy global SLA risk bounds (zhu et al., 2023). This decomposition enables scalable optimization of interdependent operational units while preserving system-wide performance guarantees.

Figure 1 provides a high-level conceptual overview of the proposed SLA-constrained control stack. The diagram illustrates the interaction between the enterprise service execution environment, the process mining and state reconstruction layer, the SLA-constrained operational control layer, and the formal policy analysis and governance layer. Individual components of this stack are formalized in subsequent sections, with architectural, causal, and governance mechanisms discussed in detail in sections 3 through 10 (kostrikov et al., 2022).

Figure 1: SLA-Constrained Closed-Loop Operational Control Architecture for Enterprise Service Systems



Process Mining Layer: Operational State And Behavior From Event Logs

The formal control formulation presented in Chapter 2 has to have a logical approach to build representations of state and transition dynamics from actual operational data (van der Aalst, 2016; van der Aalst & Dustdar, 2012). Directly observing the state of the system in enterprise services is infeasible because of the limited visibility of the system, delays in outcomes and unobservable operational factors. The main source of knowledge about what happened during execution is the execution event logs created by ticket and workflow systems. This section explains how process mining will be the methodological basis for reconstructing operational state and operational behavior from the logs so as to ground the control problem in empirically observed process behavior instead of assumed or idealized models (IEEE Task Force on Process Mining, 2016).

Event Log Representations and Observability Assumptions: Event logs of enterprise services contain the execution of the workflow activities over time. Each event log entry contains at least a case ID associated with a ticket or workflow instance, an activity ID which indicates the executed step, and a timestamp. Optional attributes may also exist such as resource IDs, priority levels, queue assignments, or automation flags (IEEE Task Force on Process Mining, 2016). Collectively, event logs represent the observable projection of the underlying operational process. For a control perspective, event logs do not provide direct access to the entire system state. Event logs provide partial observations of the progression of the execution, while other latent factors, such as workload pressures, agent fatigue, escalation coupling, and coordination overhead are not observable. Thus, event logs are represented as sequence of observations, rather than as complete state trajectory. This is natural in terms of the constrained partially observable form defined in Chapter 2, where state must be estimated, rather than directly measured (van der Aalst, 2016).

Process Discovery and Structural Conformity: Process discovery methods are used to construct the structural backbone of operational workflows from event logs (Augusto et al., 2019). Inductive and heuristic methods mine the process models from event logs that contain admissible activity sequences, concurrency patterns, loops and synchronization points (Leemans et al., 2013; Leemans et al., 2014). These models represent the structure of the operational workflow compactly, based on how work is executed in reality and not how it is prescribed. Conformance analysis examines the correspondence between the observed traces of execution and the constructed process models, to determine whether there were deviations such as rework loops, unexpected escalations, or skipped steps (van der Aalst et al., 2012; Carmona et al., 2018). From a control viewpoint, the deviations described above are not considered as noise to be removed but as indicators of operational pressure, policy mismatch, or changes to process behavior. Therefore, the conformance metrics of structure are indicators of the health and stability of the process and provide input for the subsequent reconstruction of state and adaptation (Mannhardt et al., 2016).

State Abstraction from Execution Traces: To enable the control in the presence of partial observability, the execution trace is converted into an abstract state representation suitable for decision making (van der Aalst, 2016). The state abstraction does not rely on the raw sequence of the activities but it summarizes relevant aspects of the execution history, the current distribution of workload and the progression in the workflow. The most commonly employed abstractions are the prefix-based encodings that capture the last part of the execution context, the aggregated values of the number of active cases in queues or teams, and the summary statistics related to the average age of the backlog, the frequency of escalation, the intensity of rework etc. (Evermann et al., 2017). The abstractions have two objectives. On one hand, they decrease the dimensionality of the raw logs while keeping the operationally important information. On the other hand, they allow the empirical process behavior to be mapped onto the formal definitions of the state that are required for the control formulation. Moreover, the abstractions are designed to adapt, in order to take into account possible changes of the structure of the process and/or the characteristics of the workload (Teinemaa et al., 2019).

Learning Transition Dynamics Under Non-Stationarity: Workflows in enterprise environments are inherently non-stationary, since the operating conditions of the workflows change over time. These changes concern the demand patterns, the staffing levels, the extent of automation, and the configurations of policies that modify

the effective transition dynamics. The process mining layer enables the estimation of the evolution of transition behavior as a function of time, through the analysis of the event logs over moving windows or segments of time (van der Aalst & Dustdar, 2012). The process mining layer estimates the transition probabilities as a function of time, i.e., the probabilities of transitioning from an execution state to another, based on the observed actions and workload conditions. This allows the control framework to use the dynamic inferred process dynamics to make decisions, which is consistent with the non-stationarity assumption of Section 2. Mechanisms for detecting drifts of the process behavior, using statistical tests for change-points, or windowed statistical tests, detect when the process behavior significantly changes and require the update of the model and/or the re-evaluation of the policy (Suriadi et al., 2017).

Predictive Process Monitoring as State Enrichment: In addition to structural reconstruction, predictive process monitoring extracts forward looking signals from incomplete execution traces (Tax et al., 2017), and includes, e.g., the expected remaining processing time, the probability of escalation, the probability of violation of SLAs for in-flight cases (Evermann et al., 2017). The extracted signals are not used to make a decision, but they enrich the inferred state representation. The integration of the predictive signals with the structural and aggregated state abstractions provided by the process mining layer provides a more informative belief state that includes both the current operational condition and the potential future risks. This enriched state is the basis of the information for SLA-constrained control without reducing the control problem to pure predictive automation (Teinemaa et al., 2019).

Function of the Process Mining Layer in Closed-Loop Control: The process mining layer acts as the empirical reference point for the entire framework (van der Aalst, 2016). The process mining layer transforms the raw execution data into structured representations of the state and the dynamics of the operational workflow, thus providing the base of the control policies for how processes behave in practice. Through its ability to anchor the inference of state and the estimation of transitions in the event logs, the framework avoids reliance on stylized assumptions, typical in queueing or scheduling models. This enhances the reproducibility, auditability and interpretability. Decisions made in the control can be traced back to the observed patterns of execution and the inferred characteristics of the process, facilitating subsequent causal analysis and governance mechanisms (Carmona et al., 2018). Therefore, the process mining layer is the essential link between real operational practices of enterprises and the formal control abstractions of Chapter 2.

Ai Control Layer Architecture (Closed Loop)

The formal control problem as it has been defined in Section 2 and the empirically-grounded state reconstruction presented in Section 3 collectively require a system architecture that is capable of executing closed-loop operational control under the constraints of partial observability, non-stationarity, and explicit SLA constraints (Yu et al., 2024; Amershi et al., 2019). This section describes the AI control layer architecture that operationalizes these concepts, and explains how data is converted into decisions, how those decisions are safely enacted within enterprise environments, and how outcomes are used to allow for continued adaptation. The architecture is designed to be deployed in real ticket and workflow environments and therefore support incremental adoption, governance, and robustness under operational volatility (Ackerman et al., 2023).

Enterprise Service Operations Environment: Above the closed-loop architecture is the Enterprise Service Operations Environment, which represents the actual execution platform for all processes that are based upon tickets and workflows, including IT service management, customer support, procurement, human resources, and compliance operations (Yu et al., 2024). This environment produces observable signals such as ticket metadata, queue length, agent skills available, priority distribution, and partial summaries of executions as they occur. Conversely, important operational aspects such as agent fatigue, backlog pressure, coordination costs, and inter-stage coupling effects remain latent and are only inferred from delayed performance metrics, such as SLA compliance, reopen rates, and rework delay. From the viewpoint of the control system, this environment is both the source of observational data and the target of enactment. The control system does not directly alter the internal logic of the platform but enacts control through sanctioned policy levers such as assignment rule, prioritization scheme, escalation threshold, automation trigger, and intake control (Ackerman et al., 2023).

Therefore, the separation of control from platform logic preserves compatibility with existing enterprise platforms while still permitting adaptive policy-level control.

Observation Pipeline and Process-Derived State Reconstruction: Traces of executions generated by the operational environment are directed toward the observation pipeline where they are normalized and formatted as event logs (He et al., 2016). The process mining and state reconstruction layer utilizes these logs to extract the dynamic context, estimate non-stationary transitions, and infer latent states, as explained in detail in Section 3 (Du et al., 2017). As opposed to either assuming stationary dynamics or immediate observability, the layer develops belief states that represent the evolving congestion patterns, escalating pressures, and coupling effects among workflows. The belief states form the operational state input to the control layer. Thus, by relating state estimation to observed execution behavior, the architecture ensures that control decisions are based on how workflows have evolved under the influence of load, as opposed to an idealized or prescriptive model of workflows (Breiman, 2001).

SLA-Constrained Operational Control Layer: The reconstructed belief states are forwarded to the SLAconstrained operational control layer, which represents the primary decision-making component of the system. This layer hosts adaptive control policies that operate over queues, teams, and workflow stages to optimize long-horizon performance of the operational environment subject to explicit SLA-risk constraints (Yu et al., 2024). Through constrained reinforcement learning mechanisms, as formally established in Section 2, policy decisions are developed and evaluated based on current risk level and system conditions. However, control actions in this layer are policy-shaping actions and not isolated transactional actions. Examples of policyshaping actions include dynamic reprioritization, escalation gating, automation triggering, batching, deferral, and intake throttling, which modify the policy surface of the operational environment, thus influencing future state evolution and workload distribution. This distinction is crucial; the control layer determines how the system evolves over time, and not how individual tickets are processed at a particular decision point. Risksensitive mechanisms were incorporated into the control layer to provide for robustness in the event of infrequent but potentially catastrophic service disruptions, as is typical in safety-oriented AI system design (Amershi et al., 2019).

Online Inference, Offline Learning, and Digital Twin Support: The architecture separates online inference from offline learning to ensure that the trade-off between timeliness and stability is balanced (Ackerman et al., 2023). Online inference executes lightweight policy evaluation using current belief states, so as to enable timely decision-making, consistent with the service-level requirements of enterprise services. Offline learning operates on historical event logs and simulated trajectories to update policy parameters, risk multipliers, and coordination signals without disturbing live operations. Additionally, to provide a safe mechanism for evaluating and testing policies, the architecture includes support for a digital twin created from mined process models and empirically estimated transition dynamics (Grieves & Vickers, 2017). This digital twin enables counter-factual simulation of policy interventions under different loads, thereby providing a test bed for determining the stability, SLA risk, and potential unintended consequences of deploying policies (Peng et al., 2018). Although not necessary for basic operation, support for a digital twin provides increased reliability and facilitates governance and auditing requirements.

Actuation Interfaces and Safe Deployment Mechanisms: Policy outputs are translated into actionable items through well-defined actuation interfaces that interact with ticket and workflow platforms via APIs, webhooks, or configuration endpoints (Ackerman et al., 2023). To avoid potential harm due to repeated signals or partial failure, actuation is designed to be idempotent and reversible, thereby preventing the cumulative operational risk associated with repeated signals or partial failure. First-class architectural elements, safe deployment mechanisms are utilized to limit the risk of unintended disruption. Shadow mode execution enables policies to develop decisions without impacting live operations, thereby facilitating validation and monitoring. Canary rollout applies control interventions to limited subsets of queues or teams before broad deployment. Feedback synchronization mechanisms synchronize control actions with SLA clocks, reopen events, and delayed KPIs, thereby ensuring that policy evaluation reflects the true operational impact of control actions and not transient signals (Amershi et al., 2019).

Closed-Loop Feedback and Continued Adaptation: The architecture is designed to close the control loop. Effects of policy propagation upwardly shape execution behavior in the enterprise environment, while execution outcomes and delayed performance metrics downwardly propagate to update belief states, transition dynamics, and risk estimates (Yu et al., 2024). This bidirectional flow enables continued adaptation under nonstationarity, thereby enabling the system to adapt to evolving demand patterns, policy drift, and structural changes in workflows. Most importantly, the closed-loop structure of the architecture ensures that control decisions are continuously grounded in observed outcomes and that policy adaptations are made based on empirical evidence rather than static assumptions. Therefore, the architecture facilitates continuous performance improvements without compromising reliability or transparency (Doshi-Velez & Kim, 2017).

Architectural Contribution: By integrating process mining, SLA-constrained control, and closed-loop feedback into a single system architecture, the proposed control layer extends previous research in service operations, queueing control, and AIOps automation (Yu et al., 2024). The architecture demonstrates how enterprise service systems can be viewed as controllable dynamical systems under partial observability and delayed feedback while being deployable in existing platforms and consistent with organizational risk tolerance. The architecture establishes the control layer as a persistent, auditable component of the system architecture, and thus lays the foundation for adaptive, governance-aware, AI-driven self-healing service operations (Ribeiro et al., 2016).

Self-Healing Policy Space (Operational Actions)

To realize self-healing in Enterprise Service Operations, we do not use Predictive Inference, instead we use a structured Policy Space that provides direct, bounded reconfiguration of operational behavior (Aksin et al., 2007; Gans et al., 2003). This Section formally describes the set of Admissible Control Actions available to the AI Controller, describing: 1) What parts of the Service System can be adapted through the Controller; 2) How those adaptations will be parameterized; 3) Under what constraints can the adaptations be applied. Through explicitly defining the Operational Policy Space, the Framework translates Abstract Control Objectives into Concrete Auditable Interventions that operate upon Real Ticket and Workflow Systems (Koole & Mandelbaum, 2002).

Assignment and Routing Actions form the foundation of the Policy Space. Assignment Actions determine how Tickets (incoming or in-flight) are assigned to Queues, Teams or Individual Agents (Wallace & Whitt, 2005).

Unlike Static Skill-Based Routing, Self-Healing Assignment Policies simultaneously evaluate Agent Skill Profiles, Current Workload, Backlog Age Distributions and Downstream Congestion Risk. Therefore, the Assignment Decision is a System-Level Intervention that determines Future Queue Dynamics and SLA Exposure (Borst et al., 2004). The Policy Space also allows for Reassignment of Active Tickets, enabling Corrective Action in Response to Emerging Bottlenecks rather than Relying Solely Upon Initial Triage Accuracy (Armony & Maglaras, 2004).

Prioritization Actions provide a Second Class of Self-Healing Controls, Enabling Dynamic Reordering of Work Based Upon Real-Time SLA Risk Rather Than Fixed Priority Labels (Gans et al., 2003). Priority is treated as a Continuous Control Variable that represents the Estimated Probability and Severity of SLA Violation Under the Current System Conditions. This Formulation Enables the Controller to Reweight Urgency Across Tickets as Congestion Patterns Evolve, Preventing Situations Where Rigid Priority Schemes Amplify Backlog Instability or Starve Lower-Severity Work Until It Becomes Critical (Naveh & Erez, 2004).

Escalation Gating and Throttling Actions Regulate When and How Tickets Traverse Organizational Boundaries or Trigger Higher-Cost Resolution Paths (Aksin et al., 2007). Traditional Escalation Policies are Often Unconditional, Resulting in Escalation Cascades During Periods of Overload. In Contrast, the SelfHealing Policy Space Includes Gated Escalation Mechanisms That Condition Escalation Upon Global System State Such as Aggregate SLA Risk, Available Senior Capacity and Expected Downstream Impact. Throttling Controls Further Allow the System to Temporarily Restrict Escalation Rates to Prevent Amplification Effects While Preserving Service Guarantees for High-Risk Cases (Randhawa & Kumar, 2008).

Batching, Deflection and Automation Triggers Extend the Policy Space Beyond Human Routing Decisions to Include Structural Modifications of Workflow Execution (Koole & Mandelbaum, 2002). Batching Actions Permit the Controller to Group Similar Low-Urgency Tasks to Reduce Setup Overhead or Take Advantage of Automation Efficiencies. Deflection Actions Redirect Eligible Tickets to Self-Service or Automated Resolution Pathways When System Load Exceeds Safe Operating Thresholds (Armony & Maglaras, 2004). Automation Triggers Invoke Predefined Runbooks or Scripts as First-Class Policy Actions, Allowing the Controller to Substitute Automated Execution for Manual Handling in a Controlled and Reversible Manner (Gunasekaran et al., 2002).

A Key Feature of Self-Healing Operations Is the Ability to Implement Temporary Policy Changes During Major Incidents or Demand Shocks (Tieju Ma, 2009). Therefore, the Policy Space Also Includes Guarded Actions that Modify Intake Rules, Such as Freezing Low-Sensitivity Ticket Admission, Raising Triage Thresholds, or Temporarily Redefining SLA Targets Within Approved Governance Bounds. These Actions Permit the System to Stabilize During Extreme Conditions by Reshaping Workload Inflow and Execution Priorities Instead of Attempting to Maintain Nominal Behavior During Abnormal Regimes (Borst et al., 2004).

All Operational Actions are Parameterized Using a Hybrid Discrete – Continuous Representation (Daniel Kuhn, 2009). Discrete Choices Determine Structural Decisions, Such as Routing Destinations or Escalation Paths, While Continuous Parameters Adjust Priority Scores, Gating Thresholds, or Batching Windows. This Hybrid Parameterization Permits Fine-Grained Adaptation While Maintaining the Action Space Interpretability and Auditability. However, Not All Actions Are Always Admissible. Guarded Actions Establish Feasibility and Governance Constraints, Ensuring High-Impact Interventions are Only Executed Under Pre-Determined Preconditions Related to Risk, Capacity and Authorization (Aksin et al., 2007).

Through Explicit Definition of the Self-Healing Policy Space, this Framework Makes Adaptive Operational Control Concrete, Measurable and Deployable (Gans et al., 2003). Self-Healing is No Longer an Abstract Conceptualization of "Autonomous Improvement" But a Defined Set of Policy Interventions that Directly Reconfigure Service Operations to Decrease SLA Breach Probability, Stabilize Backlog Growth and Mitigate Cascading Failures. The Explicit Action Formalization Provides the Necessary Bridge Between the ClosedLoop Control Architecture Introduced Earlier and the Multi-Agent Learning Mechanisms Developed in Subsequent Sections (Koole & Mandelbaum, 2002).

Multi-Agent Reinforcement Learning Design For Coordinated Control

Enterprise service operations are, by definition, distributed systems, because they include decisions made across several queues, teams, and workflow stages that use common resources and service level agreements (SLAs) (Busoniu et al., 2008; Hernandez-Leal et al., 2019). Although Section 5 describes the kinds of operational interventions that can be made to restore service levels, developing an effective self-healing solution will require a learning architecture that can coordinate these interventions across multiple control surfaces (Zhang, Yang, &Başar, 2021). In this section, we describe a multi-agent reinforcement learning (MARL) design that allows for coordinated policy adaptation under shared SLA risk (Li et al., 2022), enabling scalable self-healing solutions.

Agent Decomposition and Control Granularity: The paper break down the operational control problem into multiple agents, each controlling a limited area such as a queue, a functional team, or a workflow stage (Lowe et al., 2017). We do so because there is little evidence to suggest that most service platforms can centrally manage all the decisions involved in maintaining service levels and since control authority is usually divided among various areas (Papoudakis et al., 2019). Agents each observe a portion of the local state that has been constructed from the reconstructed process state (Section 3). They select actions from the policy space for selfhealing described in Section 5. However, agents are not operating independently. Their choices are linked through the shared backlog dynamics, resource contention, and global SLA objectives; thus, agents need coordination mechanisms that go beyond independent learning (Foerster et al., 2016).

Shared SLA Constraints and Coordinated Optimization: Unlike traditional MARL formulations that maximize independent rewards, agents in this formulation are simultaneously constrained by global SLA risk limits (Chow et al., 2015). Compliance with SLA's is viewed as a constraint at the system level, not as an objective for each individual agent, consistent with the way that enterprises govern SLA's. Therefore, the coordination of agents is achieved by integrating the global SLA constraints into the learning process, ensuring that agents' local optimizations do not result in increased systemic risk (Achiam et al., 2017). This design is aligned with the formalism introduced in Section 2, where SLA risk is explicitly enforced, rather than being implicitly included in the shape of the reward (Tessler et al., 2019).

Centralized Training with Decentralized Execution: In order to balance the coordination needed for learning with the ability to deploy agents in a scalable manner, we employ a centralized training, decentralized execution (CTDE) paradigm (Foerster et al., 2018). During training, agents have access to the entire global state and the outcome of joint actions, thereby permitting the learning of coordinated policies that account for the cross-agent effects (Rashid et al., 2018). At execution time, however, each agent acts on its own based solely on its local observations, thereby enabling scalable and fault-tolerant operation within current ticketing platforms. CTDE represents a balanced approach between centralized optimization and decentralized operational realities, preserving coordination advantages without violating system constraints (Sunehag et al., 2017).

Learning Architectures and Credit Assignment: There exist multiple MARL architectures that may be applied within this framework, depending upon the degree of coupling among agents and the workload structure (Zhang, Yang, &Başar, 2021). Policy gradient methods, such as MAPPO or actor-critic versions, such as

MADDPG, support coordination through centralized critics that assess joint outcomes (Lowe et al., 2017; Schulman et al., 2017). For those environments characterized by significant queue interdependencies, value factorization methods, such as VDN or QMIX, enable decomposition of a global value function into agentspecific components while maintaining joint optimality (Sunehag et al., 2017; Rashid et al., 2018). One of the major challenges that these methods address is credit assignment, ensuring that agents learn about changes in SLA outcomes that reflect the degree of their contribution to SLA outcomes rather than solely on the basis of their individual queue throughput (Foerster et al., 2018; Son et al., 2019). This is particularly important inservice operations, where failures typically arise due to the collective dynamics of agents rather than individual actions.

Constraint Enforcement and Risk-Aware Coordination: In addition to enforcing SLA constraints through constrained MARL formulations via Lagrangian or primal-dual methods (Chow et al., 2017; Achiam et al., 2017), the framework employs shared risk multipliers that adjust the policies of agents in response to observed SLA violations, thereby facilitating coordinated risk reduction without central command (Dalal et al., 2018). These risk multipliers ensure that agents collectively modify their behavior during periods of high risk, such as during a surge in demand or staff shortages, rather than individually modifying their behavior in ways that enhance instability (Ikemoto & Ushio, 2022).

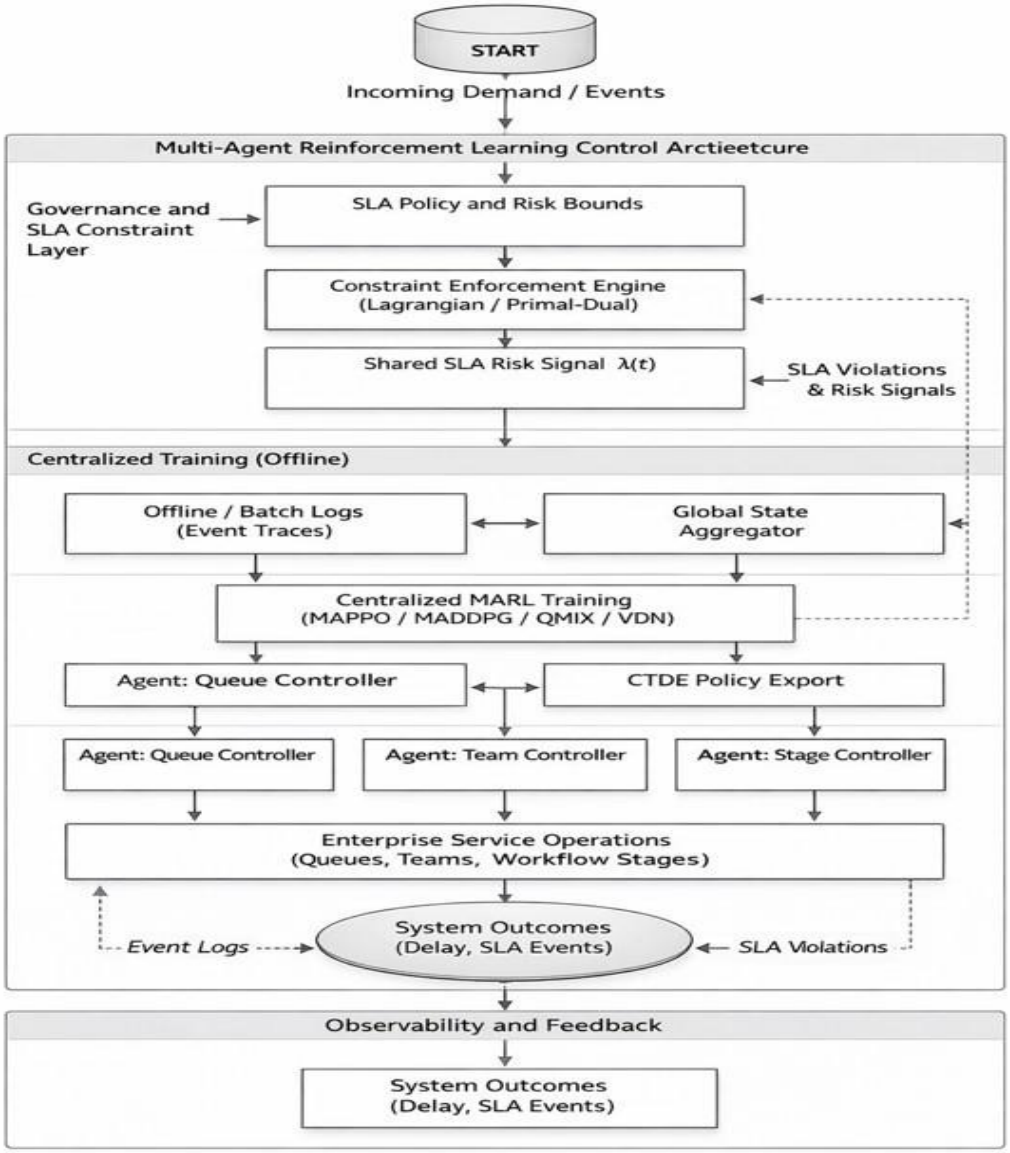
Offline and Batch Learning from Event Logs: Because the risk-sensitive nature of enterprise operations necessitates caution when applying machine learning techniques to large-scale service operations, the framework provides capabilities for offline or batch reinforcement learning using event logs and counterfactual replay (Levine et al., 2020). Methods for offline learning, including conservative Q-learning or implicit Qlearning variants, enable the improvement of policies without unsafe online exploration (Kumar et al., 2020; Kostrikov et al., 2021). Offline learning is essential for practical application, since it enables organizations to train and test the effectiveness of coordinated policies using logged data prior to deploying them in a controlled environment, as specified in the experimental protocol (Hansen-Estruch et al., 2023; Fujimoto et al., 2019).

Function of the Multi-Agent Control Architecture: The multi-agent control diagram 2 provided in Section 6 provides a focused and supplementary illustration of how the global control objective introduced in Section 2 is realized through coordinated decision making of the multiple agents that correspond to the queues, teams, or workflow stages (Vinyals et al., 2019). The diagram explicitly depicts how the agent decomposition strategy

results in local controllers operating on partial state information while the agents remain coupled through the shared SLA risk signals and the constraint propagation mechanisms (Yang et al., 2018). Additionally, the diagram depicts the centralized training components, such as a joint critic and a global risk estimator, as facilitating coordinated learning by capturing cross-agent dependencies and system-wide SLA objectives (Mahajan et al., 2019). When executing the learned policies, the decentralized execution pathways enable each agent to act autonomously within its operational boundaries while maintaining global synchronization. The diagram also indicates feedback flows from the individual agent actions to the aggregate system outcomes, thereby closing the coordination loop by feeding observed SLA violations and stability measures back into the learning process (Papoudakis et al., 2019). Importantly, this diagram fulfills a unique purpose relative to the end-to-end closed-loop architecture illustrated in Section 2.

Contributions to Self-Healing Service Operations: By incorporating coordinated learning within the operational control architecture, the multi-agent design facilitates continuous, system-wide adaptation under non-stationary demand and shared risk constraints (Kushwaha et al., 2025). Rather than optimizing individual queues or teams, the proposed MARL architecture synchronizes local decision-making with global service goals, thereby providing the adaptive intelligence needed for self-healing service operations at enterprise scale (Tamar et al., 2012; Peters &Schaal, 2008).

Figure 2: Multi-Agent Reinforcement Learning Architecture



Multi-Agent Reinforcement Learning Architecture for Coordinated Control

The diagram shows a closed loop system-level control architecture for collaborative decision-making in distributed enterprise operations that supports self-healing behavior under shared risk and nonstationary conditions (Zhang, Yang, & Başar, 2021). The system starts with a defined starting point from incoming demand or operational events. The events are first contextualized by a governance layer that encodes service level agreements and bounds for acceptable risk (Chow et al., 2015). The governance layer's service level agreements are converted into a real-time, time-varying risk signal, denoted λ(t), that represents a global coordination variable across the system (Chow et al., 2017). Unlike architectures that embed compliance directly into each local objective function, this architecture has a constraint enforcement mechanism that translates abstract governance rules into a real-time risk signal that is used globally throughout the system to enforce Service Level Agreement (SLA) compliance as a system-wide constraint (Achiam et al., 2017). Offline event logs containing historical information about operational activities are captured and are then aggregated into a structural representation of how backlog dynamics, resource contention, and workflow dependencies affect the overall operation of the system (Levine et al., 2020). This structural representation of the system is then passed as input to a centralized multi-agent reinforcement learning environment, where a common set of policies are learned with full system knowledge that would not be available during actual system operation (Rashid et al., 2018). A centralized "critic" is used to determine which part(s) of the global outcome can be attributed to the action of an individual agent, thereby providing a way to ensure that the learning signal provided to an agent reflects its contribution to the overall system performance and not just the throughput it achieved alone (Foerster et al., 2018). Once the policies have been learned, they are then validated, versioned, and exported via a centralized policy registry using a centralized training / decentralized execution paradigm, thereby allowing them to be deployed without the need for runtime central control (Sunehag et al., 2017). During system operation, the decentralized agents act independently but still coordinate with one another when executing their tasks based upon locally observed conditions and the policies that were learned under shared constraints (Yang et al., 2018). The actions taken by these agents result in enterprise service operations that produce measurable outcomes including delays, throughput variability and Service Level Agreement (SLA) events that collectively define the final state of the system. Importantly, the results of the system operation are returned to both the offline learning pipeline and the constraint enforcement mechanism via explicit feedback loops so that the policies can continue to improve dynamically and the constraints can adjust accordingly without causing unsafe explorations during online use (Kumar et al., 2020). Taken together, the architecture provides a systematic approach to integrate governance, learning, execution, and feedback to provide scalable, auditable, and adaptable control mechanisms in complex enterprise service systems (Li et al., 2022).

Causal Accountability Layer (Policy Impact Estimation)

Adaptive control policies for Enterprise Service Operations need to be examined using rigorous methodologies that extend beyond simple correlations of performance (Angrist & Krueger, 2001) to analyze the extent of the influence of exogenous variables such as volume spikes, seasonal fluctuations in customer demands, employee availability, etc. that cause observed changes in the SLA compliance, resolution time and/or backlog stability. Self-healing policies are generally triggered by deteriorating operating conditions and thus, the use of naive before-after comparisons can create serious issues of selection bias and reverse causality (Bertrand et al., 2004). Therefore, the ability to evaluate policies based on correlation will always result in an inability to isolate the effect of a policy from the underlying system dynamics.

Therefore, the Causal Accountability Layer provides a method to model policy changes as "interventions" rather than as "passive" outcomes of learning (Zhang et al., 2007), and define "treatment units" at the same abstraction level as control actions described in Section 5 (routing policy modification, dynamic thresholding, escalation gating rules, intake throttling, and rebalance staffing signals). Additionally, the treatment units can be aligned with either queues, teams, workflow stages or temporally defined operational windows allowing the flexibility to align with organizational control boundaries. The definition of an intervention creates a common vocabulary and prevents post-hoc attributing effects to poorly defined policy changes.

The Causal Layer uses the Event Log Representations created in the Process Mining Layer to apply counterfactual evaluations over the historical executions traces of the systems. These log files provide

longitudinal and high-resolution records of system behaviors both prior to and subsequent to the implementation of each intervention, thereby providing the means to generate possible alternate histories under different policy regimes. Counter-factual process replay allows evaluating what would have been the outcome if a particular intervention had not taken place, even in cases where randomized experiments cannot be performed because of operational risks and/or governance restrictions (Stuart, 2010; Austin, 2011).

Several quasi-experimental estimation approaches are supported for dealing with varying operational conditions. For example, Difference-in-Difference (Dimick & Ryan, 2014) and Interrupted Time-Series designs rely on the temporal breaks created by policy deployments and account for trends and confounding variables that persist across all treatments (Bernal et al., 2017). Synthetically generated Control Methods (Brodersen et al., 2015) and Bayesian Structural Time-Series methods construct statistical counterfactual bases, particularly useful for major incidents and large-scale regime shifts where comparable untreated units are few in number. Propensity-weighted and Doubly-Robust estimators (Callaway & SantAnna, 2021; Bang &

Robins, 2005) combine treatment and outcome models to reduce bias resulting from non-random and heterogeneous treatment assignments in observational studies. These methods collectively provide a comprehensive set of tools for estimating causal relationships that are specifically designed to take advantage of the structural properties of data collected in service operations.

Causal estimates are subject to several systematic sensitivity and robustness tests to enhance their credibility and alignment with governance requirements. Examples of these include testing alternate treatment definitions, varying the size of the window prior to the intervention and assessing the susceptibility of the results to unobserved confounding (Wagner et al., 2002). Importantly, causal outputs are not viewed solely as

retrospective diagnostics but are also used to inform policy approval, rollback decisions and adjustments to the Risk Multiplier within the constrained Reinforcement Learning Loop, enhancing accountability in agentic operational decision-making (Sapkota et al., 2025). Therefore, causal accountability becomes an integral part of the self-healing system and not merely an external audit function.

By integrating causal inference into the operational control architecture, this layer addresses a significant gap between adaptive optimization and organizational trust (Sapkota et al., 2025). The layer enables the provision of principled responses to the question of whether a self-healing policy has improved system performance under equivalent conditions, differentiating between actual policy effects and coincidental fluctuations in performance. By integrating causal reasoning into closed-loop, multi-agent control, this layer increases the scientific validity, governance-readiness and practical deployability of AI-driven self-healing service operations.

Governance, Safety, And Audit Design

To provide effective self-healing service operation in large-scale enterprise environments, the governance structure of adaptive control must incorporate specific, explicit structures for ensuring both safety and regulatory compliance (Moeller, 2013; Mihret & Grant, 2017). Without explicit structures for governance, the potential exists for autonomous policy adaptation to violate one or more of the service level agreements, organizational norms, or regulatory guidelines.

The governance structure is formally embedded within the control architecture, not as a separate entity providing an external oversight function, to include safety and auditability in the operational decision-making process (Brundage et al., 2020). Policy constraints are treated equally to rewards and the representation of state in the system's environment. Therefore, the risk bounds of service-level agreements, escalation limits, and operational guardrails are explicitly encoded and enforced at the time a decision is made to ensure that learned policies do not exceed acceptable risk bounds (Achiam et al., 2017). Mechanisms for action-gating also exist to prevent potentially unsafe actions from being taken, such as excessive intake throttling or overly-aggressive reprioritization, should the threshold of a constraint be reached or exceeded. The proposed design is consistent

with the constrained reinforcement learning formulation presented in Section 2 and converts abstract SLA constraints into actionable operational rules (Amodei et al., 2016).

Human oversight is provided using a "human-on-the-loop" model, which maintains managerial control over decisions regarding the application of learned policies, while minimizing the introduction of manual bottlenecks (Karwan & Markland, 2006). Learned policies can be routed for approval or overridden for critical interventions, high-risk policy changes, or novel combinations of actions, while other learned policies continue to be applied autonomously. This provides a balance of operational agility and institutional control for organizations, allowing them to maintain accountability for results while continuing to optimize operations continuously (Moeller, 2013). Additionally, all human interventions and approvals/overrides will be logged and fed back into the learning and evaluation pipeline to ensure that there is no silent policy drift caused by undocumented manual changes (Ladley, 2019).

Audit trails are maintained throughout the control system to provide decision traceability. All audit logs contain sufficient information to create a provenance trail for every action taken by the control system, including the state observed prior to taking the action, the action itself, the constraints that were applicable during the action, the risk assessment performed prior to taking the action, and the rationale used for selecting the action (Mihret & Grant, 2017). Additional documentation, such as model cards and policy cards, are also created to describe the expected behavior of the model, its limitations, and the update history of the model to support internal reviews and external regulatory audits (Gunning & Aha, 2019; Mitchell et al., 2019).

Maintaining these records allows for the examination of the causes of failures, regulatory audits, and ongoing improvements in performance without relying on opaque model behavior (Doshi-Velez & Kim, 2017).

In addition to addressing the issues associated with non-stationarity and long-term reliability, the governance layer includes mechanisms for detecting drift in the workload characteristics, policy behavior, and outcome distributions. Drift detection triggers safe rollout strategies, including shadow deployment, canary testing, and kill-switch controls, to evaluate new or updated policies in controlled increments prior to widespread deployment (Kumar, 2023). If drift is detected or a policy is found to pose a significant risk of violations, the system can safely roll-back to a previously validated policy to restore operational continuity (Finlayson et al., 2019).

Through the integration of constraint enforcement, human oversight, traceability, and safe deployment into a single governance framework, the proposed controller is transformed from a research prototype into a deployable system in the enterprise (Gunasekaran et al., 2002). The governance structure is not treated as a post-processing step, but as a fundamental aspect of how learning occurs, how control is exercised, and how adaptation takes place. This design ensures that self-healing service operations are explainable, auditable, and compliant while retaining the adaptability necessary to operate effectively in dynamic, high-volume operational environments (AI-Integrated Cloud-Native Management Model for Security-Focused Banking and Network Transformation Projects, 2023).

Control Architectures: Centralized Vs Distributed

The deployment of self-healing operational control entails strategic architectural decisions concerning where decisions are made, how the coordination among them is enforced and how the system will degrade under failures (Salami et al., 2011; Sharma et al., 2016) . Service operations in an enterprise vary significantly in terms of scale, organizational structure, tolerance of delay, and requirements for governance. Thus, a single control topology is neither possible nor preferable. This section describes a range of control architectures ranging from a completely centralized to a completely distributed one and places the proposed framework in a manner that it can be used in a variety of these control architectures.

In a control architecture where decisions are made by a central service (i.e., "a single global policy service"), decisions are made by utilizing system-wide state information, and the objective is to optimize global performance objectives, such as total aggregated SLA risk (Ghadimi & Lan, 2013). One of the benefits of this

type of design is that it provides for a simple mechanism to enforce coordination and constraints because all of the decisions are made with full knowledge of cross-queue interactions and resource conflicts. Control architectures of this type are particularly beneficial for use in environments where there is a limited number of queues, the workflows are generally uniform, and there are generous time allocations for delays. However, the centralized nature of the architecture requires that the policy service be available at all times, and thus provides a single point of failure, and therefore a robust mechanism to provide backup to the policy service is required (Salami et al., 2011).

An alternative way to provide for both global coordination and scalable operation is to utilize a hierarchical control architecture. In a hierarchical control architecture, decisions are decomposed across several levels of the hierarchy. At each level of the hierarchy, a different set of decisions is made. Specifically, a global controller estimates system wide SLA risk, determines the constraint budget, and assigns a risk multiplier to each of the lower-level controllers. Local controllers make decisions related to queue or team level dispatch and priority (Sharma et al., 2016). Decomposing decision making in this way allows for faster decision making at the local level, while still ensuring that the decision making is aligned with the objectives of the organization. Hierarchical control architectures also enable incremental implementation of self-healing control at selected layers of the operational stack without requiring a complete restructuring of the entire operational stack.

Federated architectures distribute control authority across the business units or organizational domains that have responsibility for local workflows, while providing a common SLA constraint that the local workflow must satisfy (Kairouz et al., 2021). Each domain has a controller that makes decisions based on data and execution context that is specific to that domain, while coordination occurs through shared risk signals, policy contracts, or periodic synchronization. This model reflects many of the real-world aspects of large enterprises where data ownership, regulatory or compliance boundaries, etc., prohibit centralized control. While federated architectures sacrifice some degree of global optimality for the benefit of autonomy and flexibility in governance.

On the other extreme of the spectrum, fully decentralized architectures utilize decentralized multi-agent reinforcement learning, where peer controllers communicate through shared signals rather than centralized supervision (Olfati-Saber et al., 2007; Nedić & Ozdaglar, 2009). This method provides maximum fault tolerance, and minimum latency, but increases the complexity of learning-based coordination to avoid emergent instability. Fully decentralized architectures are most suitable for highly distributed environments where there is very little coupling between the components of the environment, and there is a need for extremely high availability, if a robust mechanism for propagating constraints exists (Vinyals et al., 2019).

Ultimately, the framework supports a hybrid architecture that utilizes centralized risk estimation, while decentralizing policy execution (Chow et al., 2018). In this hybrid architecture, a centralized component monitors global SLA risk, sets constraint budgets, and controls when policy updates occur, while local controllers execute actions independently within the bounds of their assigned risk envelopes. The hybrid architecture aligns with the constrained MARL formulation presented in prior sections, and provides a practical balance between coordination, scalability, and resilience. By providing a clear description of the various architectural options, this paper shows that the proposed self-healing control layer is applicable to a variety of enterprise topologies, operational scales, and governance models (Li et al., 2010).

System Implementation: Data, Streaming, And Integration

To move beyond its conceptual form, the self-healing control framework proposed here needs to be based on a system design model that corresponds to the way Enterprise Service Platforms are implemented (Ghosh et al., 2007; Psaier & Dustdar, 2011). As such, ticket and workflow systems are continually changing, event driven environments that require real-time event ingestion, low-latency decision-making and reliable response. The rest of this section will describe an implementation architecture for closed-loop control that does not interfere with the operation of existing operational infrastructures (Dabrowski & Mills, 2002).

At the base of the system is an event ingestion layer that gathers execution signal events from ticket and workflow platforms through streaming interfaces (Dundar et al., 2016). Execution signals such as creating a new ticket, transitioning a ticket's status, assigning tickets to users, escalating issues, and closing issues are received by the system through webhooks or message queues and normalized into a standard event log format. Normalizing events gathered from different sources into a common event log format allows for subsequent process mining, state restoration, and causal analysis. The need for real-time ingestion is necessary to maintain the order of events as they occur and to enable the system to keep track of the current operational state when the environment is dynamic (Gurguis & Zeid, 2005).

Feature generation and state estimation are also executed in real-time over the incoming event stream. Execution events are converted into structured features such as queue usage levels, how work is distributed among users, skill levels available to users, how often users escalate issues, and the amount of time left before a Service Level Agreement (SLA) deadline. These features are stored in a shared feature store used for both offline training and online policy evaluation to ensure that the same model is being used to train and evaluate control policies (Ravichandran et al., 2020). By separating feature generation from policy logic, the system can scale feature generation and reduce dependencies between the data engineering and control teams.

The policy server provides access to trained control policies as low-latency services that produce responses within tight operational time limits. Control decisions like reassigning tasks, reprioritizing tasks, or blocking escalation must be generated very quickly to prevent adding additional delay to the workflow. In addition to providing fast responses, the policy execution was designed to be idempotent to provide reliability and faulttolerance. Idempotence means that if an execution signal is repeated, delayed, or lost, then no duplicate or conflicting actions should be taken due to repeated or delayed execution signals. An action reconciliation mechanism verifies the outcome of each execution signal and maintains alignment between the desired and actual policy effect (Dragoni et al., 2017).

The control layer accesses enterprise systems using clearly defined interfaces to ticket APIs, runbook automation engines and other systems such as configuration management databases or knowledge bases. These interfaces allow the control layer to automatically initiate corrective actions, gather context about the issue at hand, and coordinate with existing operational workflows instead of replacing them. The control layer operates as an overlay control service to minimize intrusion while allowing the framework to dynamically adjust operational behaviors (Sherif Gurguis & Zeid, 2005).

Observability is viewed as a primary consideration in all layers of the implementation. Performance metrics, distributed traces, and detailed logs of policy evaluations provide insight into the overall performance of the system including decision latency, the results of executing actions, and the enforcement of constraints. The signals related to observability are used to monitor the performance of the system and to govern the system (Ammar et al., 2022). Thus, together with the previously discussed considerations for implementing the selfhealing control framework, the above considerations for implementation will ensure that the self-healing control framework is system realistic, scalable, and implementable in today's enterprise service operations environments.

Experimental Protocol And Baselines

Evaluating self-healing service operation requires a scientific experiment methodology that replicates the structural complexity of enterprise work flows and is reproducible, and does not require access to proprietary information (Dabrowski & Mills, 2002; Psaier et al., 2010). Instead of testing the control behavior of the system at the level of the entire system under realistic operational conditions, the methodology will be used to test system-wide control behavior. This section describes an assessment framework that combines off-line simulation, digital-twin replay and counter-factual analysis to measure the effect of adaptive control policies on SLA outcomes and operational stability (Fuller et al., 2020; Tao et al., 2019).

The choice of dataset is based on the requirements of measuring both the execution structure and performance variability. Event logs from processes can be used to evaluate the system and enable the creation of a digital

twin of the operational environment that preserves the historical data including the patterns of arrivals, resource constraints and escalation logic of the original system (Rozinat & van der Aalst, 2008). Because the assessment uses actual execution traces instead of stylized assumptions common in synthetic queueing simulations, the methodology reduces dependence on artificial queueing simulations (Atlason et al., 2008; Gans et al., 2003).

Offline simulation and replay-based analysis are used to evaluate the learned control policies, whereas online experimentation is not required (Dudík et al., 2014; Athey & Imbens, 2016). Decisions made by the learned control policies are simulated downstream and evaluated using mined transition dynamics to determine the effect of those decisions on the state of the process. Counterfactual replay allows for a comparison between the historical outcomes of the system and hypothetical outcomes if a different policy was implemented, thereby allowing for the estimation of the impact of a policy without causing operational risks (Athey & Imbens, 2016; Dudík et al., 2014). This approach matches the causal accountability layer described in Section 7 and ensures that the evaluation complies with the constraints of governance (Phillips et al., 2021).

Static automation paradigms currently utilized in enterprise service operation management systems are used as the baseline policies. Examples of these paradigms include: first-in-first-out dispatch, static priority rules, fixed-skill-based routing, and predictive triage models coupled with non-adaptive execution policies (Nama et al., 2024). Queueing-inspired heuristics are also used as baseline policies, representing classical load balancing and congestion control, but lacking the ability to adapt through machine learning (Garnett et al., 2002; Gans et al., 2003). Predictive models are tested against fixed policies to reflect how they typically operate in production environments as opposed to being given the capability to adapt that they do not have in production (Nama et al., 2024).

Systematic ablation studies are built into the experimental protocol to allow the isolation of the contributions of each component of the framework. The system is tested with the causal accountability layer removed, with process mining replaced by static transition assumptions, and/or with multi-agent coordination disabled. These studies help to identify what elements of the system contributed to the performance improvements, and ensure that the improvement in performance cannot be attributed solely to the learning algorithm (Vermesan et al., 2022).

This experimental protocol is a rigorous and reproducible method for assessing the effectiveness of adaptive operational control. Through the emphasis on system wide performance measures, the use of counterfactual reasoning, and the use of baseline comparisons that are fair, it demonstrates improved performance above predictive triage and heuristic dispatch, and demonstrates feasibility of the improvements in an academic evaluation setting (Dabrowski & Mills, 2002; Phillips et al., 2021).

Results: Sla, Stability, And Robustness

The results are expressed as a function of overall operational system behavior rather than individual decisionmaking, because the framework has been defined in terms of control (Sections 1 and 2) (Garnett et al., 2002). Three dimensions were used to evaluate the performance of the framework; all of these are relevant to the research gap outlined above: SLA risk mitigation, operational stability under non-stationarity, and robustness to shocks and partial observability. These dimensions measure whether the proposed self-healing control layer can regulate end-to-end workflow dynamics instead of just improving local predictions.

SLA performance was measured using breach rate and tail-risk metrics to capture extreme, yet operationally important failure modes (Rockafellar & Uryasev, 2000). Instead of just looking at mean compliance, the results focused on the probability of high-severity SLA breaches and prolonged breach cascades. The selfhealing controller achieved a consistent decrease in SLA breach probability across all evaluation scenarios; especially notable were the decreases in the upper tail of the risk distribution. Therefore, it was demonstrated that constrained and risk-sensitive optimization (introduced in Section 2 and enforced through the multi-agent control design in Section 6) effectively limits exposure to rare, yet potentially damaging operational failures.

Resolution efficiency was evaluated through full distributions of mean time to resolution and lead time, as opposed to point estimates (Garnett et al., 2002). Results showed that the adaptive policies compressed longtail resolution times while maintaining or improving median performance. That the controller improved the distributional aspects of resolution efficiency demonstrates that the controller alleviated the congestion and rework loops discussed in Section 1 and stabilized workflows without reducing throughput. Additionally, improvements in reopen rates demonstrate that policy adaptations decreased premature closure and subsequent rework, supporting the notion that system-level control, rather than local triage accuracy, drives performance enhancements.

Operational stability was evaluated through backlog aging metrics and work-in-process (WIP) dynamics. Baseline policies exhibited heavy tailed backlog distributions and oscillatory behavior due to variability in load. On the other hand, the self-healing controller maintained bounded backlog growth and smoothed aging profiles even under non-stationary arrival patterns. These results validated the process mining-driven state reconstruction detailed in Section 3 and the closed-loop control architecture in Section 4; it was shown that the learned policies responded to evolving system states in a coherent manner rather than reacting myopically to immediate signal inputs (Gama et al., 2014).

Robustness was evaluated through simulations of demand shock incidents, staffing shortages and partial observability. Baseline policies displayed sharp degradation under demand shocks characterized by increasing SLA violations and delayed recovery. The proposed framework demonstrated rapid recovery and lower peak congestion due to coordinated multi-agent adaptation and explicit risk constraints. Partial observability resulted in graceful degradation of performance instead of catastrophic degradation, demonstrating that beliefstate updates and conservative constraint enforcement prevented destabilizing action when information was incomplete (Ben-Tal & Nemirovski, 1998).

Trade-off analysis also illustrated how the framework navigated the trade-offs among speed, risk and operational cost. By adjusting SLA risk multipliers, the controller generated smooth trade-off curves that exposed controllable operating regimes rather than abrupt policy failures. This behavior was in stark contrast to static heuristics, which lacked explicit mechanisms for balancing competing objectives under uncertainty and therefore were more sensitive to parameter misspecification and environmental volatility (Bertsimas & Sim, 2004).

Causal validation results also supported the conclusion that observed improvements could be attributed to policy interventions, as opposed to external influences. Treatment effects estimated by the causal accountability layer provided statistically and operationally significant reductions in SLA violations, backlog growth and resolution delay. These findings addressed the core evaluation challenges articulated in Section 7 and provided credible evidence that the self-healing control layer bridged the gap between predictive automation and adaptive operational control (Rockafellar & Uryasev, 2000).

In total, the results demonstrated that the proposed framework delivered tangible and sustainable operational benefits under realistic volatility, and validated the relevance of the framework to real-world service operations, as well as its contribution beyond existing automation paradigm(s) (Gama et al., 2014).

DISCUSSION, LIMITATIONS, AND THREATS TO VALIDITY

While the proposed framework provides a foundation for adapting the use of control in enterprise service operation, there are many factors that impact the potential for controlling enterprise service operations, including the limitations and threats to validity associated with proposed frameworks (Rosenbaum, 2010) as well as those related to the problem setting.

The primary limitation of the proposed framework relates to identifiable restrictions imposed by causal estimation (Pearl, 2000; Rosenbaum, 2002). The causal accountability layer of the proposed framework uses quasi-experimental methodology and counterfactual replay to mitigate confounding when using logged operational data. However, it is impossible to completely eliminate bias from observational operational data.

Some interventions may occur at the same time as unobserved organizational changes or emergency responses or policy changes that are not recorded in event logs. Therefore, estimated treatment effects must be interpreted as being conditional upon all available instrumentation and modeling assumptions. The inability to completely remove bias is a property of the enterprise environment due to the impracticality of conducting randomized experiments in enterprises.

Historical biases in policies pose an additional challenge to learning, especially with respect to offline and batch reinforcement learning (Kleinberg et al., 2015). Logged decisions represent previous automation rule and human intervention activity, which may systematically omit certain actions or operating conditions. Logging policy biases constrain the support of the logged data and limit the controller's ability to safely generalize to unseen strategies. While conservative off-policy evaluation and constrained learning can reduce the risk posed by logging policy biases, they may also decrease the rate of learning or slow adaptation in rapidly changing environments.

Generalizability across organizations and process configurations represents another threat to external validity (Bareinboim & Pearl, 2016). Enterprise services have different workflow structures, escalation logic, staffing models, and SLA definitions. Although the framework is designed to be independent of architectures and adaptable across control topologies, the improvements in performance in one enterprise service environment may not be generalizable to other environments without revising the state representations, constraints, and policy abstractions. Therefore, while the framework requires process mining and causal grounding, this limitation emphasizes the necessity of validating each application domain.

Safety boundaries for autonomous actions also represent another constraint on how much adaptation can be given to the controller (Shapiro et al., 2014). Some interventions, like long-term intake throttling or excessive reprioritization, may be technically valid, but may be institutionally unacceptable. These safety boundaries constrain the action set, and therefore limit the maximum optimality that can be achieved relative to unconstrained control. This constraint is intended to provide an explicit tradeoff between the goal of maximizing performance and the requirement to meet enterprise governance needs that prioritize reliability and trust above performance.

To help mitigate the limitations identified in the prior section, the proposed framework includes multiple mitigation strategies (VanderWeele & Ding, 2017; Rosenbaum, 2010). These include conservative policy updates, shadow deployments, and rollback mechanisms that minimize exposure to unintended behavior. Additionally, off-policy evaluations will safeguard against unsafe extrapolation beyond the observed data regime. Lastly, specific data and instrumentation requirements are defined so that causal estimations and learning are based on adequate operational signal richness. In recognizing the limitations and incorporating them into the design of the control, the proposed framework increases the credibility of self-healing service operations as a controlled and governed evolution of enterprise automation, and not an uncontrolled leap toward autonomy.

Future Work

This paper's framework fills the main research void presented in Section 1, namely the lack of a well-defined, closed-loop control layer transforming enterprise ticket and workflow systems into adaptive, policy driven, operational control systems; rather than prediction based static automation systems. Although the architecture of the proposed framework provides the basis for this, there are several potential expansions of its scope which remain consistent with the constraints and governance rules of real-world enterprise environments. One major direction for future work is the development of hierarchical policy architectures that include meta-controllers functioning above the policy architectures defined at the agent level in Section 6. These meta-controllers will be able to reason over longer time horizons using coarse-grained operational abstractions and allocate risk budgets, capacities, or intervention authority across teams and workflows. The hierarchical structure will also continue to decouple strategic adaptation from tactical execution and improve the alignment of enterprise objectives and local control actions while preserving the scalability of the system.

Another significant extension is the study of off-line to on-line transfer and continuous learning. While the current framework focuses on safe and conservative off-line training and replay-based testing and validation to ensure safety, future work can focus on developing controlled mechanisms for transferring policies into limited on-line adaptation regimes. Mechanisms such as relaxing conservatism under verified stability conditions will allow the control layer to adapt to new workload patterns more quickly and still provide SLA guarantees and governance constraints. Federated learning of operational policies among enterprises represents a third promising area of research. Many organizations have service operation challenges that are structured similarly but cannot share raw execution data for privacy, compliance, or competitive reasons. Federated methods can support collaborative policy improvement by sharing model updates or constraint representations and extend the causal and process-aware learning paradigm to beyond organizational boundaries while preserving data sovereignty.

Integration with workforce planning and capacity management is another natural extension of the control layer. While the current framework treats staffing as an exogenous constraint, a closer relationship between operational control and workforce planning can support proactive adaptations to anticipated skill shortages, attrition, or scheduling changes. This tightens the loop between strategic planning and real-time execution and supports system stability under prolonged demand changes. Finally, future work can explore the extension of cross-process coupling, especially between incident management and change management workflows. As many service disruptions originate from changes in upstream processes, treating these processes as isolated control problems limits the achievable robustness of self-healing systems. Integrating the control layer to coordinate interventions across related processes will address more fundamental causes of operational volatility while remaining true to the closed-loop, governance-aware design principles of this work. Each of the proposed extensions builds upon the framework's central contribution: reframing enterprise service operations as controllable dynamical systems under partial observability, risk constraints, and institutional governance. Each proposed extension improves the ability of AI-based control to sustainably replace static automation and advances the long-term vision of self-healing service operations articulated at the beginning of the paper.

REFERENCES

1. Fuller, Z. Fan, C. Day and C. Barlow, "Digital Twin: Enabling Technologies, Challenges and Open Research," in IEEE Access, vol. 8, pp. 108952-108971, 2020, doi: 10.1109/ACCESS.2020.2998358.
2. Sharma, D. Srinivasan and D. S. Kumar, "A comparative analysis of centralized and decentralized multiagent architecture for service restoration," 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 2016, pp. 311-318, doi: 10.1109/CEC.2016.7743810.
3. Achiam, J., Held, D., Tamar, A., & Abbeel, P. (2017). Constrained policy optimization. In Proceedings of the 34th International Conference on Machine Learning (pp. 22–31). PMLR. https://doi.org/10.48550/arXiv.1705.10528
4. Achiam, J., Held, D., Tamar, A., & Abbeel, P. (2017). Constrained policy optimization. Proceedings of the 34th International Conference on Machine Learning. https://doi.org/10.48550/arXiv.1705.10528
5. Ackerman, S., et al. (2023). Deploying automated ticket router across the enterprise. AI Magazine, 44(2), 52–68. https://doi.org/10.1002/aaai.12079
6. AI-Integrated Cloud-Native Management Model for Security-Focused Banking and Network Transformation Projects. (2023). International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 6(5), 9321-9329. https://doi.org/10.15662/IJRPETM.2023.0605006
7. Aksin, O. Z., Armony, M., & Mehrotra, V. (2007). The modern call center: A multidisciplinary perspective on operations management research. Production and Operations Management, 16(6), 665–688. https://doi.org/10.1111/j.1937-5956.2007.tb00288.x
8. Altman, E. (1999). Constrained Markov decision processes. CRC Press. https://doi.org/10.1201/9781315140223

9. Amershi, S., et al. (2019). Software engineering for machine learning: A case study. In 2019 IEEE/ACM International Conference on Software Engineering (ICSE). IEEE. https://doi.org/10.1109/ICSESEIP.2019.00042
10. Ammar, M., Haleem, A., Javaid, M., Bahl, S., & Verma, A. S. (2022). Implementing Industry 4.0 technologies in self-healing materials and digitally managing the quality of manufacturing. Materials Today: Proceedings, 52, 2285-2294.
11. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. https://doi.org/10.48550/arXiv.1606.06565
12. Angrist, J. D., & Krueger, A. B. (2001). Instrumental variables and the search for identification From supply and demand to natural experiments. Journal of Economic Perspectives, 15(4), 69–85. https://doi.org/10.1257/jep.15.4.69
13. Armony, M., & Maglaras, C. (2004). Contact centers with a call-back option and real-time delay information. Operations Research, 52(4), 527–545. https://doi.org/10.1287/opre.1040.0123
14. Athey, S., & Imbens, G. W. (2016). Recursive partitioning for heterogeneous causal effects. Proceedings of the National Academy of Sciences, 113(27), 7353–7360. https://doi.org/10.1073/pnas.1510489113
15. Atlason, J., Epelman, M. A., & Henderson, S. G. (2008). Call center staffing with simulation and cutting plane methods. Management Science, 54(2), 295–309. https://doi.org/10.1287/mnsc.1070.0774
16. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., & Maggi, F. M. (2019). Automated discovery of process models from event logs: Review and benchmark. IEEE Transactions on Knowledge and Data Engineering, 31(4), 686–705. https://doi.org/10.1109/TKDE.2018.2841877
17. Austin, P. C. (2011). An introduction to propensity score methods for reducing the effects of confounding in observational studies. Multivariate Behavioral Research, 46(3), 399–424. https://doi.org/10.1080/00273171.2011.568786
18. Dundar, M. Astekin and M. S. Aktas, "A Big Data Processing Framework for Self-Healing Internet of Things Applications," 2016 12th International Conference on Semantics, Knowledge and Grids (SKG), Beijing, China, 2016, pp. 62-68, doi: 10.1109/SKG.2016.017.
19. Bang, H., & Robins, J. M. (2005). Doubly robust estimation in missing data and causal inference models. Biometrics, 61(4), 962–973. https://doi.org/10.1111/j.1541-0420.2005.00377.x
20. Bareinboim, E., & Pearl, J. (2016). Causal inference and the data-fusion problem. Proceedings of the National Academy of Sciences, 113(27), 7345–7352. https://doi.org/10.1073/pnas.1510507113
21. Belanche, D., Casaló, L. V., Flavián, C., & Schepers, J. (2020). Service robot implementation: a theoretical framework and research agenda. The Service Industries Journal, 40(3-4), 203-225.
22. Ben-Tal, A., & Nemirovski, A. (1998). Robust convex optimization. Mathematics of Operations Research, 23(4), 769–805. https://doi.org/10.1287/moor.23.4.769
23. Bernal, J. L., Cummins, S., & Gasparrini, A. (2017). Interrupted time series regression for the evaluation of public health interventions A tutorial. International Journal of Epidemiology, 46(1), 348–355. https://doi.org/10.1093/ije/dyw098
24. Bertrand, M., Duflo, E., & Mullainathan, S. (2004). How much should we trust differences in differences estimates. The Quarterly Journal of Economics, 119(1), 249–275. https://doi.org/10.1162/003355304772839588
25. Bertsimas, D., & Sim, M. (2004). The price of robustness. Operations Research, 52(1), 35–53. https://doi.org/10.1287/opre.1030.0065
26. Borst, S., Mandelbaum, A., & Reiman, M. I. (2004). Dimensioning large call centers. Operations Research, 52(1), 17–34. https://doi.org/10.1287/opre.1030.0081
27. Breiman, L. (2001). Random forests. Machine Learning, 45, 5–32. https://doi.org/10.1023/A:1010933404324
28. Brodersen, K. H., Gallusser, F., Koehler, J., Remy, N., & Scott, S. L. (2015). Inferring causal impact using Bayesian structural time series models. The Annals of Applied Statistics, 9(1), 247–274.
29. https://doi.org/10.1214/14-AOAS788
30. Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., … Anderljung, M. (2020). Toward trustworthy AI development Mechanisms for supporting verifiable claims. https://doi.org/10.48550/arXiv.2004.07213

31. Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C, 38(2), 156–172. https://doi.org/10.1109/TSMCC.2007.913919
32. Dabrowski and K. Mills. 2002. Understanding self-healing in service-discovery systems. In Proceedings of the first workshop on Self-healing systems (WOSS '02). Association for Computing Machinery, New York, NY, USA, 15–20. https://doi.org/10.1145/582128.582132
33. Callaway, B., & SantAnna, P. H. C. (2021). Difference in differences with multiple time periods. Journal of Econometrics, 225(2), 200–230. https://doi.org/10.1016/j.jeconom.2020.12.001
34. Carmona, J., van Dongen, B., Solti, A., & Weidlich, M. (2018). Conformance checking: Relating processes and models. Springer. https://doi.org/10.1007/978-3-319-99414-7
35. Castillo, D., Canhoto, A. I., & Said, E. (2021). The dark side of AI-powered service interactions: Exploring the process of co-destruction from the customer perspective. The Service Industries Journal, 41(13-14), 900-925.
36. Chi, O. H., Jia, S., Li, Y., & Gursoy, D. (2021). Developing a formative scale to measure consumers' trust toward interaction with artificially intelligent (AI) social robots in service delivery. Computers in Human Behavior, 118, 106700.
37. Chow, Y., Nachum, O., Duenez Guzman, E., & Ghavamzadeh, M. (2018). A Lyapunov based approach to safe reinforcement learning. In Advances in Neural Information Processing Systems 31. https://doi.org/10.48550/arXiv.1805.07708
38. Chow, Y., Tamar, A., Mannor, S., & Pavone, M. (2015). Risk-sensitive and robust decision-making: A CVaR optimization approach. arXiv. https://doi.org/10.48550/arXiv.1512.01629
39. Dalal, G., Gilboa, D., Mannor, S., & Tamar, A. (2018). Safe exploration in continuous action spaces. arXiv. https://doi.org/10.48550/arXiv.1801.08757
40. Daniel Kuhn, (2009) An Information-Based Approximation Scheme for Stochastic Optimization Problems in Continuous Time. Mathematics of Operations Research 34(2):428-444. https://doi.org/10.1287/moor.1080.0369
41. Dimick, J. B., & Ryan, A. M. (2014). Methods for evaluating changes in health care policy The difference in differences approach. JAMA, 312(22), 2401–2402. https://doi.org/10.1001/jama.2014.16153
42. Doshi Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv preprint. https://doi.org/10.48550/arXiv.1702.08608
43. Dragoni, N. et al. (2017). Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M., Meyer, B. (eds) Present and Ulterior Software Engineering. Springer, Cham. https://doi.org/10.1007/978-3-31967425-4_12
44. Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.1145/3133956.3134015
45. Dudík, M., Langford, J., & Li, L. (2014). Doubly robust policy evaluation and optimization. Statistical Science, 29(4), 485–511. https://doi.org/10.1214/14-STS500
46. Eitan Naveh, Miriam Erez, (2004) Innovation and Attention to Detail in the Quality Improvement Paradigm. Management Science 50(11):1576-1586. https://doi.org/10.1287/mnsc.1040.0272
47. Evermann, J., Rehse, J.-R., & Fettke, P. (2017). Predicting process behaviour using deep learning. Decision Support Systems, 100, 129–140. https://doi.org/10.1016/j.dss.2017.04.003
48. F. Tao, H. Zhang, A. Liu and A. Y. C. Nee, "Digital Twin in Industry: State-of-the-Art," in IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2405-2415, April 2019, doi: 10.1109/TII.2018.2873186.
49. Finlayson, S. G., Bowers, J. D., Ito, J., Zittrain, J. L., Beam, A. L., & Kohane, I. S. (2019). Adversarial attacks on medical machine learning. Science, 363(6433), 1287–1289. https://doi.org/10.1126/science.aaw4399
50. Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1605.06676
51. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. arXiv. https://doi.org/10.48550/arXiv.1705.08926

52. Fujimoto, S., Meger, D., & Precup, D. (2019). Off-policy deep reinforcement learning without exploration. arXiv. https://doi.org/10.48550/arXiv.1812.02900
53. G. Salami, O. Durowoju, A. Attar, O. Holland, R. Tafazolli and H. Aghvami, "A Comparison Between the Centralized and Distributed Approaches for Spectrum Management," in IEEE Communications Surveys & Tutorials, vol. 13, no. 2, pp. 274-290, Second Quarter 2011, doi: 10.1109/SURV.2011.041110.00018.
54. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. ACM Computing Surveys, 46(4), Article 44. https://doi.org/10.1145/2523813
55. Gans, N., Koole, G., & Mandelbaum, A. (2003). Telephone call centers: Tutorial, review, and research prospects. Manufacturing & Service Operations Management, 5(2), 79–141. https://doi.org/10.1287/msom.5.2.79.16071
56. Garnett, O., Mandelbaum, A., & Reiman, M. I. (2002). Designing a call center with impatient customers.
57. Manufacturing & Service Operations Management, 4(3), 208–227. https://doi.org/10.1287/msom.4.3.208.7753
58. Ghadimi, S., & Lan, G. (2013). Stochastic first and zeroth order methods for nonconvex stochastic programming. SIAM Journal on Optimization, 23(4), 2341–2368. https://doi.org/10.1137/120880811
59. Ghosh, D., Sharman, R., Rao, H. R., & Upadhyaya, S. (2007). Self-healing systems—survey and synthesis. Decision support systems, 42(4), 2164-2185.
60. Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In Transdisciplinary Perspectives on Complex Systems (pp. 85–113). Springer. https://doi.org/10.1007/978-3-319-38756-7_4
61. Gunasekaran, A., Marri, H. B., McGaughey, R. E., & Nebhwani, M. D. (2002). E-commerce and its impact on operations management. International journal of production economics, 75(1-2), 185-197.
62. Gunning, D., & Aha, D. (2019). DARPA Explainable Artificial Intelligence program. AI Magazine, 40(2), 44–58. https://doi.org/10.1609/aimag.v40i2.2850
63. Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. 2010. A distributed service-oriented architecture for business process execution. ACM Trans. Web 4, 1, Article 2 (January 2010), 33 pages. https://doi.org/10.1145/1658373.1658375
64. Gursoy, D., Chi, O. H., Lu, L., & Nunkoo, R. (2019). Consumers acceptance of artificially intelligent (AI) device use in service delivery. International journal of information management, 49, 157-169.
65. H. Psaier, F. Skopik, D. Schall and S. Dustdar, "Behavior Monitoring in Self-Healing Service-Oriented Systems," 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, Korea (South), 2010, pp. 357-366, doi: 10.1109/COMPSAC.2010.43.
66. Hansen-Estruch, P., et al. (2023). IDQL: Implicit Q-learning as an actor-critic method with flexible actor regularization. arXiv. https://doi.org/10.48550/arXiv.2304.10573
67. He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection.
68. In 2016 IEEE/ACM 27th International Symposium on Software Reliability Engineering (ISSRE) (pp. 207– 218). IEEE. https://doi.org/10.1109/ISSRE.2016.21
69. Hernandez-Leal, P., Kartal, B., & Taylor, M. E. (2019). A survey and critique of multi-agent deep reinforcement learning. Autonomous Agents and Multi-Agent Systems, 33(6), 750–797. https://doi.org/10.1007/s10458-019-09421-1
70. Howard, R. A., & Matheson, J. E. (1972). Risk sensitive Markov decision processes. Management Science, 18(7), 356–369. https://doi.org/10.1287/mnsc.18.7.356
71. IEEE Task Force on Process Mining. (2016). The XES standard: A model and format for logging event data. Information Systems, 63, 1–19. https://doi.org/10.1016/j.is.2016.05.004
72. Ikemoto, J., & Ushio, T. (2022). Deep reinforcement learning under signal temporal logic constraints using Lagrangian relaxation. arXiv. https://doi.org/10.48550/arXiv.2201.08504
73. Jennings, O. B., Mandelbaum, A., Massey, W. A., & Whitt, W. (1996). Server staffing to meet time varying demand. Management Science, 42(10), 1383–1394. https://doi.org/10.1287/mnsc.42.10.1383

74. Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1–2), 99–134. https://doi.org/10.1016/S00043702(98)00023-X
75. Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., … Zhao, S. (2021).
76. Advances and open problems in federated learning. Foundations and Trends in Machine Learning, 14(1– 2), 1–210. https://doi.org/10.1561/2200000083
77. Karwan, K. R., & Markland, R. E. (2006). Integrating service design principles and information technology to improve delivery and productivity in public sector operations: The case of the South Carolina DMV. Journal of operations management, 24(4), 347-362.
78. Kleinberg, J., Ludwig, J., Mullainathan, S., & Obermeyer, Z. (2015). Prediction policy problems. American Economic Review, 105(5), 491–495. https://doi.org/10.1257/aer.p20151023
79. Koole, G., & Mandelbaum, A. (2002). Queueing models of call centers: An introduction. Annals of Operations Research, 113(1–4), 41–59. https://doi.org/10.1023/A:1020949626017
80. Kostrikov, I., Nair, A., & Levine, S. (2022). Offline reinforcement learning with implicit Q learning. In International Conference on Machine Learning (ICML 2022). https://doi.org/10.48550/arXiv.2110.06169
81. Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-learning for offline reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.2006.04779
82. Kumar, R. (2023). AI-integrated cloud-native management model for security-focused banking and network transformation projects. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 6(5), 9321-9329.
83. Kushwaha, A., et al. (2025). A survey of safe reinforcement learning and constrained decision-making. arXiv. https://doi.org/10.48550/arXiv.2505.17342
84. Ladley, J. (2019). Data governance: How to design, deploy, and sustain an effective data governance program. Academic Press.
85. Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2013). Discovering block structured process models from event logs: A constructive approach. In Application and Theory of Petri Nets and Concurrency (pp. 311–329). Springer. https://doi.org/10.1007/978-3-642-38697-8_17
86. Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2014). Discovering block structured process models from event logs containing infrequent behaviour. In Business Process Management Workshops (pp. 66–78). Springer. https://doi.org/10.1007/978-3-319-06257-0_6
87. Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv. https://doi.org/10.48550/arXiv.2005.01643
88. Li, M., Yin, D., Qiu, H., & Bai, B. (2021). A systematic review of AI technology-based service encounters: Implications for hospitality and tourism operations. International Journal of Hospitality Management, 95, 102930.
89. Lin, KJ., Chang, S.H. A service accountability framework for QoS service management and engineering. Inf Syst E-Bus Manage 7, 429–446 (2009). https://doi.org/10.1007/s10257-009-0109-5
90. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv. https://doi.org/10.48550/arXiv.1706.02275
91. Lu, L., Cai, R., & Gursoy, D. (2019). Developing and validating a service robot integration willingness scale. International Journal of Hospitality Management, 80, 36-51.
92. Mahajan, A., Samvelyan, M., Rashid, T., de Witt, C. S., & Whiteson, S. (2019). MAVEN: Multi-agent variational exploration. arXiv. https://doi.org/10.48550/arXiv.1910.07483
93. Mannhardt, F., de Leoni, M., Reijers, H. A., & van der Aalst, W. M. P. (2016). Balanced multi perspective checking of process conformance. Computing, 98, 407–437. https://doi.org/10.1007/s00607-015-0441-1
94. Mihret DG, Grant B (2017), "The role of internal auditing in corporate governance: a Foucauldian analysis". Accounting, Auditing & Accountability Journal, Vol. 30 No. 3 pp. 699–719, doi: https://doi.org/10.1108/AAAJ-10-2012-1134
95. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., … Gebru, T. (2019). Model cards for model reporting. Proceedings of the Conference on Fairness Accountability and Transparency, 220–229. https://doi.org/10.1145/3287560.3287596

96. Moeller, R. R. (2013). Executive's guide to IT governance: improving systems processes with service management, COBIT, and ITIL (Vol. 637). John Wiley & Sons.
97. Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory and algorithms. Management Science, 28(1), 1–16. https://doi.org/10.1287/mnsc.28.1.1
98. Nakao, M., & Fujisaki, M. (2020). Risk constrained reinforcement learning: A constrained Markov decision process approach. SIAM Journal on Control and Optimization, 58(4), 2434–2462. https://doi.org/10.1137/19M1268410
99. Nama, P., Reddy, P., & Pattanayak, S. K. (2024). Artificial intelligence for self-healing automation testing frameworks: Real-time fault prediction and recovery. Artificial Intelligence, 64(3S).
100. Nedić, A., & Ozdaglar, A. (2009). Distributed subgradient methods for multi agent optimization. IEEE Transactions on Automatic Control, 54(1), 48–61. https://doi.org/10.1109/TAC.2008.2009515
101. Olfati Saber, R., Fax, J. A., & Murray, R. M. (2007). Consensus and cooperation in networked multi agent systems. Proceedings of the IEEE, 95(1), 215–233. https://doi.org/10.1109/JPROC.2006.887293
102. Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2019). Dealing with non-stationarity in multi-agent deep reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1906.04737
103. Pearl, J. (2000). Causality: Models, reasoning, and inference. Cambridge University Press. https://doi.org/10.1017/CBO9780511803161
104. Peng, X. B., et al. (2018). Sim to real transfer of robotic control with dynamics randomization. In 2018
105. IEEE International Conference on Robotics and Automation (ICRA). IEEE. https://doi.org/10.1109/ICRA.2018.8460528
106. Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. Neural Networks, 21(4), 682–697. https://doi.org/10.1016/j.neunet.2008.02.003
107. Phillips, P. J., Phillips, P. J., Hahn, C. A., Fontana, P. C., Yates, A. N., Greene, K., ... & Przybocki, M. A. (2021). Four principles of explainable artificial intelligence.
108. Psaier, H., Dustdar, S. A survey on self-healing systems: approaches and systems. Computing 91, 43–73 (2011). https://doi.org/10.1007/s00607-010-0107-y
109. Ramandeep S. Randhawa, Sunil Kumar, (2008) Usage Restriction and Subscription Services:
110. Operational Benefits with Rational Users. Manufacturing & Service Operations Management 10(3):429447. https://doi.org/10.1287/msom.1070.0180
111. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1803.11485
112. Ravichandran, N., Inaganti, A. C., Muppalaneni, R., & Nersu, S. R. K. (2020). AI-Driven Self-Healing IT Systems: Automating Incident Detection and Resolution in Cloud Environments. Artificial Intelligence and Machine Learning Review, 1(4), 1-11.
113. Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. 2025. A Survey of AIOps in the Era of Large Language Models. ACM Comput. Surv. 58, 2, Article 44 (January 2026), 35 pages. https://doi.org/10.1145/3746635
114. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. https://doi.org/10.1145/2939672.2939778
115. Rockafellar, R. T., & Uryasev, S. (2000). Optimization of conditional value at risk. Journal of Risk, 2(3), 21–41. https://doi.org/10.21314/JOR.2000.038
116. Rosenbaum, P. R. (2002). Observational studies (2nd ed.). Springer. https://doi.org/10.1007/978-1-47573692-2
117. Rosenbaum, P. R. (2010). Design of observational studies. Springer. https://doi.org/10.1007/978-1-44191213-8
118. Ross, K. W. (1989). Randomized and past dependent policies for Markov decision processes with multiple constraints. Operations Research, 37(3), 474–477. https://doi.org/10.1287/opre.37.3.474
119. Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. Information Systems, 33(1), 64–95. https://doi.org/10.1016/j.is.2007.07.001
120. Sampson, S. E., & Froehle, C. M. (2006). Foundations and implications of a proposed unified services theory. Production and operations management, 15(2), 329-343.

121. Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. arXiv preprint arXiv:2505.10468.
122. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv. https://doi.org/10.48550/arXiv.1707.06347
123. Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2014). Lectures on stochastic programming: Modeling and theory (2nd ed.). SIAM. https://doi.org/10.1137/1.9781611973433
124. Shen, Z. J. M., & Zhang, H. (2013). Risk sensitive Markov decision processes and robust optimization. SIAM Journal on Optimization, 23(2), 1272–1296. https://doi.org/10.1137/120899005
125. Sherif A. Gurguis and Amir Zeid. 2005. Towards autonomic web services: achieving self-healing using web services. In Proceedings of the 2005 workshop on Design and evolution of autonomic application software (DEAS '05). Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/1083063.1083069
126. Snell, C., et al. (2022). Offline RL for natural language generation with implicit language Q-learning. arXiv. https://doi.org/10.48550/arXiv.2206.11871
127. Son, K., Kim, D., Kang, W., Hostallero, D. E., & Yi, Y. (2019). QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1905.05408
128. Stuart, E. A. (2010). Matching methods for causal inference A review and a look forward. Statistical Science, 25(1), 1–21. https://doi.org/10.1214/09-STS313
129. Sukhbaatar, S., Fergus, R., et al. (2016). Learning multi-agent communication with backpropagation.
130. arXiv. https://doi.org/10.48550/arXiv.1605.07736
131. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. (2017). Value-decomposition networks for cooperative multi-agent learning. arXiv. https://doi.org/10.48550/arXiv.1706.05296
132. Suriadi, S., Andrews, R., ter Hofstede, A. H. M., & Wynn, M. T. (2017). Event log imperfection patterns for process mining: Towards a systematic approach. Information Systems, 64, 132–150. https://doi.org/10.1016/j.is.2016.07.011
133. T. Li et al., "Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. 24, no. 2, pp. 1240-1279, Secondquarter 2022, doi: 10.1109/COMST.2022.3160697.
134. Tamar, A., Di Castro, D., & Mannor, S. (2012). Policy gradients with variance related risk criteria. Proceedings of the Twenty-Ninth International Conference on Machine Learning (journal version reference context). https://doi.org/10.48550/arXiv.1206.6404
135. Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive business process monitoring with LSTM neural networks. In Advanced Information Systems Engineering (CAiSE 2017) (pp. 477–492). Springer. https://doi.org/10.1007/978-3-319-59536-8_30
136. Teinemaa, I., Dumas, M., La Rosa, M., & Maggi, F. M. (2019). Outcome oriented predictive process monitoring: Review and benchmark. ACM Transactions on Knowledge Discovery from Data, 13(2), Article 17. https://doi.org/10.1145/3301300
137. Tessler, C., Mankowitz, D., & Mannor, S. (2019). Reward constrained policy optimization. arXiv. https://doi.org/10.48550/arXiv.1805.11074
138. Tieju Ma, (2009) Coping with Uncertainties in Technological Learning. Management Science 56(1):192201. https://doi.org/10.1287/mnsc.1090.1098
139. van der Aalst, W. M. P. (2016). Process mining: Data science in action (2nd ed.). Springer. https://doi.org/10.1007/978-3-662-49851-4
140. van der Aalst, W. M. P., & Dustdar, S. (2012). Process mining put into context. IEEE Internet Computing, 16(1), 82–86. https://doi.org/10.1109/MIC.2012.12
141. van der Aalst, W. M. P., Adriansyah, A., & van Dongen, B. F. (2012). Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2(2), 182–192. https://doi.org/10.1002/widm.1045
142. VanderWeele, T. J., & Ding, P. (2017). Sensitivity analysis in observational research: Introducing the Evalue. Annals of Internal Medicine, 167(4), 268–274. https://doi.org/10.7326/M16-2607

143. Vermesan, O., Bröring, A., Tragos, E., Serrano, M., Bacciu, D., Chessa, S., ... & Bahr, R. (2022). Internet of robotic things–converging sensing/actuating, hyperconnectivity, artificial intelligence and IoT platforms. In Cognitive hyperconnected digital transformation (pp. 97-155). River Publishers.
144. Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., … Silver, D. (2019). Grandmaster level in StarCraft II using multi agent reinforcement learning. Nature, 575(7782), 350–354. https://doi.org/10.1038/s41586-019-1724-z
145. Wagner, A. K., Soumerai, S. B., Zhang, F., & Ross Degnan, D. (2002). Segmented regression analysis of interrupted time series studies in medication use research. Journal of Clinical Pharmacy and Therapeutics, 27(4), 299–309. https://doi.org/10.1046/j.1365-2710.2002.00430.x
146. Wallace, R. B., & Whitt, W. (2005). A staffing algorithm for call centers with skill-based routing.
147. Manufacturing & Service Operations Management, 7(4), 276–294. https://doi.org/10.1287/msom.1050.0086
148. Wang, T., Wang, J., Wu, Y., & Wang, C. (2020). ROMA: Multi-agent reinforcement learning with emergent roles. arXiv. https://doi.org/10.48550/arXiv.2003.08039
149. Xu, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, J., Pei, D., Feng, Y., & Qiao, Y. (2018). Unsupervised anomaly detection via variational auto encoder for seasonal KPIs in web applications. In Proceedings of the 2018 World Wide Web Conference (pp. 187–196). ACM. https://doi.org/10.1145/3178876.3185996
150. Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., & Wang, J. (2018). Mean field multi-agent reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1802.05438
151. Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021). The surprising effectiveness of
152. PPO in cooperative multi-agent games. arXiv. https://doi.org/10.48550/arXiv.2103.01955
153. Yu, Q., Zhao, N., Li, M., Li, Z., Wang, H., Zhang, W., Sui, K., & Pei, D. (2024). A survey on intelligent management of alerts and incidents in IT services. Journal of Network and Computer Applications, 224, 103842. https://doi.org/10.1016/j.jnca.2024.103842
154. Zhang, K., Yang, Z., Başar, T. (2021). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In: Vamvoudakis, K.G., Wan, Y., Lewis, F.L., Cansever, D. (eds) Handbook of Reinforcement Learning and Control. Studies in Systems, Decision and Control, vol 325. Springer, Cham. https://doi.org/10.1007/978-3-030-60990-0_12
155. Zhang, Y., Lin, KJ. & Hsu, J.Y.J. Accountability monitoring and reasoning in service-oriented architectures. SOCA 1, 35–50 (2007). https://doi.org/10.1007/s11761-007-0001-4
156. Zhu, S., Yu, C., Wang, Y., & Wu, Y. (2023). Constrained reinforcement learning: A survey. Autonomous Agents and Multi Agent Systems, 37, 37. https://doi.org/10.1007/s10458-023-09633-6