Reinforcement Learning-Driven Predictive Maintenance Planning via Timed Automata Modeling

Ferhat Tamssaouet
LAAS-CNRS, Université de Toulouse, France

Pauline Ribot
LAAS-CNRS, Université de Toulouse, France

Yannick Pencolé
LAAS-CNRS, CNRS, Université de Toulouse, France

Abstract

Predictive maintenance of multi-component systems requires decisions that account for both local component degradation and system-level redundancy. This paper proposes a formal framework for synthesizing interpretable maintenance policies from a system functional architecture. The system is modeled as a network of Dynamic Priced Timed Game Automata (DPTGA): each component automaton encodes degradation dynamics, stochastic failure windows, and repair and replacement costs, while a System Health Monitor automaton is automatically derived from the minimal cut sets of the functional architecture and detects system failure as soon as any cut set becomes unavailable. We execute the DPTGA semantics in an explicit-state simulator and optimize a compact threshold policy with the Cross-Entropy Method (CEM). The policy contains one age-fraction threshold per component and an embedded subsystem-safe predicate that forbids any preventive action that would immediately realize a cut set, making the learned strategy provably safe and directly interpretable as a rule-based maintenance plan. On a seven-component benchmark comprising a parallel subsystem and a 2-out-of-5 redundant block, the CEM policy reduces mean cumulative cost by 15% and median cost by 16% relative to the best rule-based heuristic (corrective, systematic, and condition-based maintenance), while maintaining a 98.7% system survival rate. The approach therefore provides a compact and explainable alternative to deep reinforcement learning for architecture-aware predictive maintenance.

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks; Computing methodologies → Reinforcement learning; Software and its engineering → Model-driven software engineering

Keywords and phrases System Remaining Useful Life, Predictive Maintenance, Timed Automata, Reinforcement Learning

Digital Object Identifier 10.4230/OASIcs.DX.2026.12

1 Introduction

Maintaining complex systems requires a proactive approach to ensure reliability, minimize downtime, and extend the operational lifespan of the whole system. There are several types of maintenance. Corrective maintenance consists in restoring the system to its normal condition after a failure without any kind of anticipation. With such maintenance strategy, the system always has unexpected downtime and due to the type of failure, the restoring time might not be anticipated which drastically decreases the level of reliability of the system. To avoid such problems, preventive maintenance consists of a set of planned actions to avoid failures and extend the lifespan of the system, such plans can be systematic (e.g. actions performed after a fixed and regular amount of time) or conditional (based on some system conditions). With preventive maintenance, action plans are scheduled in advanced with anticipated costs

to minimize downtime however such strategy can be costly as the planned action, like a replacement action, might not be necessary. Predictive maintenance is going one step forward and aims at scheduling repair or replacement actions before the occurrence of any failure by performing an estimation of the Remaining Useful Life (RUL) based on the use of health indicators that collect measurements at operating time. At the component-based level, the estimation of RUL usually relies on physical measurements and degradation or ageing models that are either obtained by expertise or by machine learning techniques. At the system-based level, the objective is to estimate the System Remaining Useful Life (SRUL). In complex systems like aircrafts [16], semiconductor manufacturing systems [22], railways systems [11], the estimation of the SRUL cannot simply be summarized as the minimal RUL on one of its equipment as these systems are composed of redundant components ensuring that if one is failing the system still operates. Therefore, the SRUL not only depends on the RUL of its components but also on the functional structure of the system [29].

This paper addresses the problem of determining a predictive maintenance strategy for such complex systems based on the functional structure of the system and a real-time estimation of the components' RULs. We first propose the definition of a formal architecture based on *Dynamic Priced Time Game Automata* (DPTGA) that models the set of possible maintenance strategies with their respective costs. The architecture is firstly expressed through functional minimal cut sets, which may be extracted from a fault tree of the system, a reliability block diagram, or another dependability model. These cut sets are then used to build a network of DPTGA embedded with some maintenance/repair costs. Component automata model degradation clocks, stochastic failure windows, repair and replacement durations, and costs. A System Health Monitor (SHM) synchronizes with component events and detects system failure whenever a cut set becomes unavailable. We then propose a first method to search for a maintenance strategy that relies on a reinforcement-learning-based policy search. More specifically, this policy is obtained by simulating this DPTGA network and optimizing a compact threshold policy with the Cross-Entropy Method (CEM). The central design choice is to keep the learned policy interpretable. Instead of learning an unconstrained neural controller, the policy contains one preventive-maintenance threshold per component and a system-aware safety predicate that forbids preventive actions that would immediately realize a cut set. This makes the learned strategy easy to inspect and directly usable as a rule-based maintenance plan.

The paper is organized as follows. Section 2 positions the contribution with respect to timed-automata-based maintenance modeling, reinforcement learning for predictive maintenance, and learning-augmented controller synthesis. Section 3 introduces the necessary background on fault trees, minimal cut sets and dynamic priced timed game automata. Section 4 presents the DPTGA models for the components and the SHM. Section 5 formulates the cost-minimization problem and the CEM policy search. Section 6 reports the case study and discusses the experimental results and the learned policy. Section 7 concludes and outlines directions for future work.

2 Related Work

The proposed framework lies at the intersection of three threads: (i) timed-automata-based maintenance modeling, (ii) reinforcement learning for multi-component predictive maintenance, and (iii) learning-augmented controller synthesis on timed game automata.

Timed Automata for Fault-tree-driven Maintenance

The closest line of work translates fault trees and their extensions into networks of priced timed automata and uses Statistical Model Checking (SMC) to estimate dependability metrics. Ruijters and Stoelinga's *Fault Maintenance Trees* [33, 32] extend industry-standard fault trees with preventive maintenance and repair templates, translate them to UPPAAL priced timed automata and use UPPAAL-SMC [6] to estimate reliability, mean-time-to-failure and maintenance cost on railway and smart-building case studies. Kumar et al. [14] introduce *Attack-Fault-Maintenance Trees* (AFMTs), a safety-security hybrid that extends AFTs [15] with proactive maintenance and reactive repair, again analyzed via SMC. In a broader dependability perspective, Rubin and Du [30] review Dynamic Fault Tree (DFT) analysis for complex redundant systems with capacity degradation, showing that DFTs can capture sequence-dependent failures, reconfiguration, and redundancy, while still facing scalability issues and difficulties in representing partial degradation states. These works provide rich formal modeling frameworks for maintenance analysis, but the maintenance strategy remains a manual exploration of a finite set of templates or scenarios, the optimization does not exploit online per-component RUL information that Prognostics and Health Management (PHM) pipelines now make available.

Reinforcement Learning for Multi-component Maintenance

Several recent works formulate predictive maintenance as a sequential decision problem and solve it with optimization or (deep) reinforcement learning techniques. Regattieri et al. [28] propose a simulation-based framework for optimizing aircraft maintenance policies by combining failure-process modeling, repair-process modeling, and Monte Carlo evaluation of preventive/corrective maintenance mixes. Their objective minimizes a total service cost including corrective, preventive, inspection, and spare-part stock terms, but these costs are fixed parameters rather than degradation-dependent quantities. Reinforcement-learning (RL) approaches go further by learning state-dependent policies: Andriotis and Papakonstantinou [3] couple constrained Partially Observable Markov Decision Processes (POMDPs) with multi-agent Deep Reinforcement-learning (DRL) to derive inspection-and-maintenance policies for deteriorating infrastructure under budget and risk constraints. Najafi and Lee [23] embed a proportional-hazards degradation model into a DRL agent for multi-unit repair-based maintenance. Liu et al. [19] address multi-component systems with complex structure, and Phuc et al. [7] use multi-agent DRL for systems with multi-dependent components. Pinciroli et al. [25, 26] demonstrate that PPO with imitation pre-training significantly outperforms scheduled and condition-based baselines on wind-farm operation and maintenance equipped with PHM signals. Two recent surveys [24, 20] catalogue state designs, reward shaping, and open issues in scalability, interpretability, and certification. However, these approaches typically rely on hand-coded transition kernels, simulation models, or discrete-time Markov Decision Process (MDP) abstractions, and do not derive the decision environment from a *by-construction* guarantee that the synthesized policy respects a system-level safety predicate.

Coupling RL with PTGA: Model-checking versus Simulation

A complementary line couples reinforcement learning with verification engines that operate on (Priced) Timed Game Automata (PTGA). UPPAAL Stratego [5] synthesizes near-optimal strategies for stochastic priced timed games by combining symbolic exploration with reinforcement learning, and has been instantiated for online controller synthesis in floor heating [17],

energy-aware buildings [1], STOchastic Model-Predictive Control (STOMPC) [9] and, more recently, discretized Euclidean MDPs [12]. While model-checking-based RL provides strong formal guarantees, it inherits the well-known state-space explosion of dense-time verification: the cost per training episode grows steeply with the number of components and clocks, which is precisely the regime targeted by predictive maintenance of multi-component systems [10]. To the best of our knowledge, no previous work has reported a population-based policy search on a PTGA model of a fault-tree-derived multi-component maintenance problem with RUL-dependent dynamic maintenance costs.

Positioning of this paper

We retain the PTGA modeling language of the SMC/Stratego line – locations, clocks, guards, MCS-derived health monitor, costs – but switch from symbolic exploration to an explicit-state simulator written from the formal semantics of the model. This trade-off sacrifices on-line model-checking guarantees in exchange for a simulation throughput that is compatible with population-based policy search; the safety properties that the synthesized policy must satisfy are then enforced *by construction* via a parameterized policy class with an embedded subsystem-safe predicate (Section 5). Compared with the DRL line, we trade representational power for interpretability and sample efficiency by restricting the policy to a compact threshold class with one parameter per component, optimized by the CEM [31]. The result is a pipeline that maps a fault tree to a deployable maintenance policy without requiring either a model-checking back-end or a deep neural network in the inner loop.

3 Preliminaries

3.1 Failure Prognostics

Failure prognostics aims to estimate the future health state of a component from observable quantities such as sensor measurements, usage profiles, environmental conditions, or degradation indicators. The central output of a prognostic model is the RUL, defined as the time remaining before a component can no longer fulfill its intended function. Depending on the approach, the RUL can be delivered as a point estimate, a probability distribution, or a confidence interval, each carrying different implications for decision-making under uncertainty [4, 37]. Prognostic methods broadly fall into three families: physics-based models, which exploit degradation equations derived from domain knowledge; data-driven models, which learn degradation patterns from historical run-to-failure data; and hybrid approaches that combine both.

While most of the prognostics literature addresses single-component systems, industrial assets are inherently multi-component, and the health of one subsystem can influence or accelerate the degradation of others. System-level prognostics therefore extends individual RUL estimation to the joint health state of an assembly, accounting for load sharing, failure propagation, and structural and functional dependencies [35].

In the present work, each component i is characterized by a stochastic failure deadline sampled uniformly from the interval $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$, which represents the prognostic uncertainty window produced by an upstream health-monitoring module. The maintenance policy does not observe this deadline directly; it only has access to the degradation clock c_i and the publicly available RUL bounds. This formulation captures a realistic PHM scenario in which prognostic models constrain the failure window without revealing the exact failure instant, thereby requiring the policy to reason and act under uncertainty.

3.2 Minimal Cut Sets, Fault Trees

The proposed framework handles multi-component systems with complex architectural configurations (see for instance the functional architecture of components in Figure 4). In such an architecture, an *event* is the occurrence of a failure in a part of the system, and among them *elementary events* (also called *basic events*) are failures at the component level. Our proposed framework relies on *Minimal Cut Sets* (MCS for short) that are minimal sets of elementary events whose occurrence implies the whole system's failure. Minimal cut sets can be automatically extracted for instance from *fault trees*. Fault tree analysis is a top-down, deductive failure analysis method that uses a boolean logic tree structure to represent the elementary events and their combinations leading to the occurrence of an undesirable state (called the top event), representing system failure [18]. The combinations of elementary events are modeled using standard logic gates (e.g., AND, OR, k-out-of-n, XOR). Fault tree analysis is a very common analysis method that is available for many existing or future systems in almost all sectors of activity [18]. There are several algorithms that can be used to obtain minimal cut sets, such as MOCUS (Method of Obtaining CUt Sets) [8], BDD (Binary Decision Diagrams) [27], etc. The proposed framework only relies on minimal cut sets. While a fault tree approach can be used for representing the system architecture, it is not necessary, similar approaches can be leveraged within the same framework, such as reliability/functional block diagrams, bond graphs, etc., as long as minimal cut sets can be extracted from them.

▶ **Example 1.** The functional architecture of the system that is studied throughout this paper is illustrated in Figure 4. It is a system with 7 components with functional redundancies for high reliability.

- **Subsystem 1 (Components A and B):** A parallel redundancy where both components A and B must fail simultaneously to trigger a system-level failure (i.e., a logic AND gate).
- **Subsystem 2 (Components C, D, E, F, G):** A k-out-of-n configuration. In this setup, the subsystem fails if more than 3 components fail (i.e., 4 out of 5 components become unavailable). This implies the system requires at least two of these five components to remain operational to maintain system integrity.

In this example, there are 6 minimal cut sets $\mathcal{K}_1, \ldots, \mathcal{K}_6$: $\mathcal{K}_1 = \{A, B\}$, $\mathcal{K}_2 = \{C, D, E, F\}$, $\mathcal{K}_3 = \{C, D, E, G\}$, $\ldots$, $\mathcal{K}_6 = \{D, E, F, G\}$.

3.3 Dynamic Priced Timed Game Automata and Strategies

The maintenance optimization framework that we propose relies on a set of *Dynamic Priced Timed Game Automata* (DPTGA for short). DPTGA is an extension of the classical Timed Automata from [2].

▶ **Definition 2** (Dynamic Priced Timed Game Automata). *A* Dynamic Priced Timed Game Automaton $\mathcal{G}$ *is a tuple*

$$(L, X, \ell_0, \Sigma, C, E, \text{inv}, f)$$

where L *is a finite set of* locations, X *is a set of variables* x *over a domain* $Dom(x)$, $\ell_0 \in L$ *is the* initial location, $\Sigma = \Sigma_c \cup \Sigma_u$ *is the set of* actions *(partitioned into* controllable *and* uncontrollable *actions),* C *is a finite set of real-valued* clocks, $E \subseteq L \times \text{Guard}(C) \times \Sigma \times 2^C \times 2^X \times L$ *is a finite set of* transitions, $\text{inv} : L \mapsto \text{Guard}(C)$ *maps an* invariant *to each location,* $f : E \times \mathbb{R}^{|C|} \to \mathbb{R}$ *maps a time dependent* cost function *to each transition.*

As any timed automaton, a DPTGA is a finite-state automaton enriched with a finite set of real-valued clocks. Guards and invariants are conjunctions of atomic constraints of the form $c_1 \sim v$ or $c_1 - c_2 \sim v$ with $c_1, c_2 \in C$, $v \in \mathbb{R} \cup X$, and $\sim \in \{<, \leq, =, \geq, >\}$. The first extension provided by a DPTGA is that the set of actions is split into two disjoint sets (controllable and uncontrollable actions) so that the DPTGA is considered as a two-player game (an agent which is able to decide controllable actions from Σ_c in interaction with its environment that can produce uncontrollable actions from Σ_u). The second extension is that locations L and transitions E are associated with a cost function f that determines the cost of triggering the transition depending on the current values of the clocks. The third extension is the possibility to update variables $x := v, v \in Dom(x)$ in a transition, similarly as a clock.

The semantics of a DPTGA is a *Dynamic Priced Timed Transition System* (DPTTS) over states (ℓ, v_C, v_X) with $\ell \in L$, clock valuations $v_C : C \to \mathbb{R}_{\geq 0}$, variable valuations $v_X : X \to Dom(X)$ satisfying $\text{inv}(\ell)$. Transitions are either (i) *time elapse* (labelled $(\ell, v_C, v_X) \xrightarrow{d} (\ell, v_C{+}d, v_X)$), advancing all clocks by a non-negative delay d while continuously satisfying the invariant, or (ii) *discrete* edges (labelled $(\ell, v_C, v_X) \xrightarrow{a} (\ell', v'_C, v'_X)$) resulting from the fire of a transition $(\ell, g, a, R_C, R_X, \ell') \in E$ with $(v_C, v_X) \models g$, performing action $a \in \Sigma$, resetting clocks in R_C, updating variables in R_X, and moving to ℓ' with $v'_C = v_C[R_C] \wedge v'_X = v_X[R_X] \models \text{inv}(\ell')$. A *run* $\alpha \in \text{Runs}(\mathcal{G})$ is a finite sequence of transitions $s_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$ from the underlying DPTTS starting at state $s_0 = (\ell_0, \{0, \ldots, 0\}, \{v_{x1}, \ldots, v_{xm}\})$. The cost of α is defined as:

$$\text{cost}(\alpha) = \sum_{\substack{(\ell_{i-1}, v_{Ci-1}, v_{Xi-1}) \xrightarrow{a_i} (\ell_i, v_{Ci}, v_{Xi}), \\ a_i \in \Sigma, \\ t_i = (\ell_i, g, a_i, R_C, R_X, \ell_{i+1}) \in E}} f(t_i, v_{Ci-1}). \tag{1}$$

The proposed framework relies on the composition of DPTGA's defined here below. As detailed in Section 4, the objective is to determine an optimal maintenance strategy for a single agent which implies that any controllable (resp. uncontrollable) action at the component level is a controllable (resp. uncontrollable) action at the system level. Moreover, the global cost of a shared action (if any) is the sum of the costs.

▶ **Definition 3** (Composition of DPTGA). *Let* $\mathcal{G}_i = (L_i, X_i, \ell_{0,i}, \Sigma_{ci} \cup \Sigma_{ui}, C_i, E_i, \text{inv}_i, f_i), i \in \{1, 2\}$ *be a couple of DPTGA, the composition* $\mathcal{G}_1 \| \mathcal{G}_2$ *is the DPTGA* $\mathcal{G} = (L, X, \ell_0, \Sigma, C, E, \text{inv}, f)$ *such that:*

- $L = L_1 \times L_2$*;*
- $X = X_1 \cup X_2$*;*
- $\ell_0 = (\ell_{0,1}, \ell_{0,2})$*;*
- $\Sigma_u = \Sigma_{u1} \cup \Sigma_{u2}$, $\Sigma_c = \Sigma_{c1} \cup \Sigma_{c2}$ *with* $\Sigma_u \cap \Sigma_c = \varnothing$, $\Sigma = \Sigma_u \cup \Sigma_c$*;*
- $C = C_1 \cup C_2$*;*
- E *is the set of transitions:*
  - synchronized transitions*:* $((l_1, l_2), g_1 \wedge g_2, a, R_{C1} \cup R_{C2}, R_{X1} \cup R_{X2}, (l'_1, l'_2))$ *if* $(l_1, g_1, a, R_{C1}, R_{X1}, l'_1) \in E_1$ *and* $(l_2, g_2, a, R_{C2}, R_{X2}, l'_2) \in E_2$*;*
  - asynchronous transitions*:* $((l_1, l_2), g_1, a_1, R_{C1}, R_{X1}, (l'_1, l_2))$ *if* $(l_1, g_1, a_1, R_{C1}, R_{X1}, l'_1) \in E_1$ *and* $a_1 \notin \Sigma_2$, $((l_1, l_2), g_2, a_2, R_{C2}, R_{X2}, (l_1, l'_2))$ *if* $(l_2, g_2, a_2, R_{C2}, R_{X2}, l'_2) \in E_2$ *and* $a_2 \notin \Sigma_1$*;*
- $\text{inv}(l_1, l_2) = \text{inv}_1(l_1) \wedge \text{inv}_2(l_2)$*;*
- $f = f_1 + f_2$.

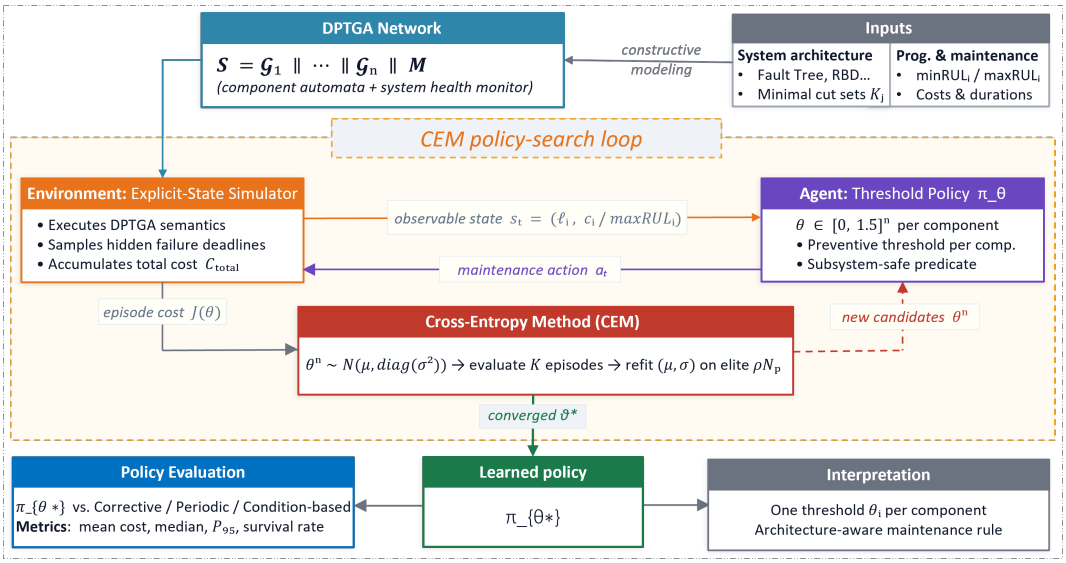

Figure 1 Overview of the proposed DPTGA-based maintenance optimization framework.

A *strategy* is a policy $\pi : \text{Runs}(\mathcal{G}) \to \Sigma_c$ over the possible runs α of the DPTGA that associates a set of possible controllable actions from Σ_c to perform with each possible run α. A strategy π is said to be *memoryless* if the set of controllable actions only depends on the last state s of the run ($\pi(s) \in \Sigma_c$). A *winning strategy* is a strategy that ensures any run respecting the strategy reaches one location of a set *Goal* (a run respects a strategy π if, in any state of the run, one of the controllable actions proposed by the strategy π (if not empty) is performed by the run). Such a strategy is said to be *optimal* if it leads to runs with a minimal cost $cost(\alpha)$.

3.4 Reinforcement Learning

Reinforcement Learning (RL) formalizes sequential decision-making through the framework of a Markov Decision Process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}(s' \mid s, a)$ the transition kernel, $\mathcal{R}(s, a)$ the reward function, and $\gamma \in [0, 1]$ a discount factor. An agent observes the current state s_t, selects an action a_t according to a policy $\pi(a \mid s)$, transitions to s_{t+1}, and receives a scalar reward r_t. The objective is to find a policy π^* that maximizes the expected discounted return $\mathbb{E}_\pi\left[\sum_{t=0}^{T} \gamma^t r_t\right]$.

RL algorithms are broadly divided into value-based methods, which estimate the optimal action-value function $Q^*(s, a)$ and derive a greedy policy from it (e.g., Q-learning [36], DQN [21]), and policy-gradient methods, which directly optimize the parameters of a parameterized policy π_θ using gradient estimates (e.g., REINFORCE, PPO [34]).

When the policy class is smooth and the parameter space is low-dimensional, gradient-free population-based methods offer a compelling alternative. The CEM [31] is one such approach: it maintains a distribution over policy parameters, iteratively samples a population of candidate policies, retains the elite fraction that achieves the highest return, and refits the distribution to these elites. CEM avoids the variance issues of stochastic gradient estimators and is particularly well-suited to maintenance scheduling problems, where the action space is combinatorial but the policy can often be compactly parameterized. In this work, we rely on CEM to optimize the maintenance policy described in Section 5.

4 Modeling

This section presents the formal modeling of the set of possible predictive maintenance strategies of a multi-component system. It provides a constructive mapping from architectural minimal cut sets (see Section 3.2) to a network of dynamic priced timed game automata (see Section 3.3) composed of component models and a SHM.

4.1 Component Modeling

In the proposed modeling framework, we assume that the system is monitored by a Health Management System that is composed of a set of prognostic agents (one per component). The prognostic agent for component i is in charge of estimating at a given time t the RUL of the component by providing a couple of pessimistic and optimistic RULs, namely minRUL_i and maxRUL_i.

Each component i is then formalized as a DPTGA $\mathcal{G}_i = \langle L_i, X_i, \ell_{0i}, \Sigma_i, C_i, E_i, \mathrm{inv}_i, f_i \rangle$ with locations $L_i = \{\mathsf{faultless}_i, \mathsf{faulty}_i, \mathsf{repair}_i, \mathsf{replacement}_i\}$ and initial location $\ell_{0i} = \mathsf{faultless}_i$ (see Figure 2). Location $\mathsf{faultless}_i$ models the fact that the prognostic agent considers the component has no fault and its remaining useful life is greater or equal to minRUL_i. Location faulty_i models the health of the component after the occurrence of the fault that is estimated to be within time interval $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$ from the initial location. Location $\mathsf{replacement}_i$ models the state of the component at the time when a corrective maintenance action is performed, that is the full replacement of the faulty component. In this model, we also might expect that the component can be fixed (location repair_i) before the complete failure of the component. DPTGA $\mathcal{G}_i$ handles two clocks $C_i = \{c_i, r_i\}$. Clock c_i tracks the component's degradation level. Initialized with $c_i = 0$ in the initial state, it ensures that the health state becomes faulty with the RUL estimation $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$ if no repair action $\mathsf{beginRep}_i$ is performed before. This behavior is modeled in $\mathcal{G}_i$ by the invariant $\mathrm{inv}(\mathsf{faultless}_i) = c_i \leq \mathsf{maxRUL}_i$ (which enforces exiting the $\mathsf{faultless}$ location before exceeding the component's absolute maximum lifespan) combined with the guard $c_i \geq \mathsf{minRUL}_i$ on the transition from $\mathsf{faultless}_i$ to faulty_i (which allows this transition only when clock c_i exceeds the pessimistic RUL estimation). This clock is reinitialized ($c_i := 0$) every time the component leaves the replacement or repair locations to become once again faultless. Clock r_i tracks the duration of maintenance activities. The repair duration is $t_{\mathrm{rep},i}$ and the replacement duration is $t_{\mathrm{repl},i}$[1]. Each component contributes to a shared variable $X_i = \{Down\}$, a set of component indices initialized to $Down = \varnothing$. Whenever component i enters location faulty_i or repair_i, index i is added to $Down$; whenever it returns to $\mathsf{faultless}_i$, index i is removed. Components in $\mathsf{replacement}_i$ already carry index i in $Down$ (set upon entering faulty_i) and release it only upon returning to $\mathsf{faultless}_i$. Hence $Down$ always reflects the set of currently unavailable components.

The action space Σ_i is partitioned into controllable actions $\Sigma_{ci} = \{\mathsf{beginRep}_i, \mathsf{replace}_i\}$, representing maintainer decisions, and uncontrollable actions $\Sigma_{ui} = \{\mathsf{failure}_i, \tau_i\}$. Action $\mathsf{failure}_i$ is stochastic: it is fired by the environment when the degradation clock c_i enters the uncertainty window $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$, reflecting the fact that the exact failure instant is hidden from the agent. Action τ_i is autonomous: it fires deterministically once the maintenance clock r_i has elapsed the prescribed duration ($t_{\mathrm{rep},i}$ for a repair, $t_{\mathrm{repl},i}$ for a replacement), returning the component to $\mathsf{faultless}_i$ with both clocks reset to zero. The transition dynamics integrates the cost model described below.

[1] For simplicity, maintenance durations are fixed constants; in general they can be represented by intervals.

- **Failure (uncontrollable):** $t_1 = \mathsf{faultless}_i \xrightarrow{\mathsf{failure}_i} \mathsf{faulty}_i$ represents the failure of the component i. The associated cost models the loss of residual value through a linear depreciation:

$$f_i(t_1, \{c_i, r_i\}) = C_{\text{fail}}(c_i) = \left| C_{\text{comp},i} - \frac{C_{\text{comp},i}}{\mathsf{maxRUL}_i} c_i \right|. \tag{2}$$

  where $C_{\text{comp},i}$ represents the initial component cost. This implies that if the system fails before reaching its maximum RUL, a proportional fraction of its asset value is lost. The current formulation assumes a linear decrease of asset value over time and uniform failure probability inside the RUL interval. However, the DPTGA framework readily accommodates non-linear degradation profiles and alternative reliability distributions.
- **Preventive maintenance (controllable):** $t_2 = \mathsf{faultless}_i \xrightarrow{\mathsf{beginRep}_i} \mathsf{repair}_i$ sends the component for repair while it is still faultless, setting $r_i := 0$ and inserting i into *Down*. It incurs the cost defined as follows:

$$f_i(t_2, \{c_i, r_i\}) = C_{\text{rep}}(c_i) = C_{\text{fixed\_rep},i} + C_{\text{rate},i}\, t_{\text{rep},i} + C_{\text{comp},i}\left(1 - \frac{c_i}{\mathsf{maxRUL}_i}\right), \tag{3}$$

  which separates a fixed logistic overhead $C_{\text{fixed\_rep},i}$, the cost $C_{\text{rate},i}$ (typically the labor cost) proportional to the repair duration $t_{\text{rep},i}$, and the residual asset value lost when scrapping a component that still had useful life remaining. This decomposition ensures that preventive maintenance is not strictly dominated by corrective maintenance: a repair executed early (small c_i) recovers more residual value and thus costs more than a late repair, restoring the intuitive ordering "preventive cheaper when executed near the end of life, corrective cheaper when failure was imminent anyway." The cost remains linear in c_i and is directly usable as a DPTGA transition cost.
- **Corrective replacement (controllable):** $t_3 = \mathsf{faulty}_i \xrightarrow{\mathsf{replace}_i} \mathsf{replacement}_i$ initiates the corrective replacement procedure, resetting the maintenance clock ($r_i := 0$). It incurs the cost

$$f_i(t_3, \{c_i, r_i\}) = C_{\text{repl},i} = C_{\text{comp},i} + C_{\text{fixed\_repl},i}, \tag{4}$$

  where $C_{\text{comp},i}$ is the procurement cost of a new component and $C_{\text{fixed\_repl},i}$ is the logistic overhead of the replacement operation. Once the replacement clock reaches $r_i \geq t_{\text{repl},i}$, the autonomous action τ_i fires and returns the component to $\mathsf{faultless}_i$, resetting both clocks and sampling a new hidden failure deadline.

4.2 Generating the System Model from the Fault Tree

The SHM is a DPTGA $\mathcal{M}$ that tracks the system-level operational status by reading the shared variable *Down*, maintained by the component automata as described in Section 4.1. Recall that after each component transition, *Down* contains the indices of all components that are currently unavailable (i.e., in location faulty_i, repair_i, or $\mathsf{replacement}_i$). The SHM has two locations: operational (initial) and failed (absorbing). It stays in operational as long as no minimal cut set is entirely contained in *Down*, encoded by the location invariant

$$\text{inv}(\mathsf{operational}) = \forall j : \mathcal{K}_j \not\subseteq \mathit{Down}.$$

Whenever a component transition causes *Down* to cover a complete cut set, the SHM synchronizes on the triggering event e and either loops in operational if the invariant is still satisfied, or fires the autonomous τ-transition to the absorbing failed location when $\exists j : \mathcal{K}_j \subseteq \mathit{Down}$ (see Figure 3). Upon entering failed, a system-failure cost C_{SF} is incurred, augmented by an early-failure penalty $C_{\text{SF}}^{\text{ef}}$ if the collapse occurs before a prescribed horizon T_{ef}.

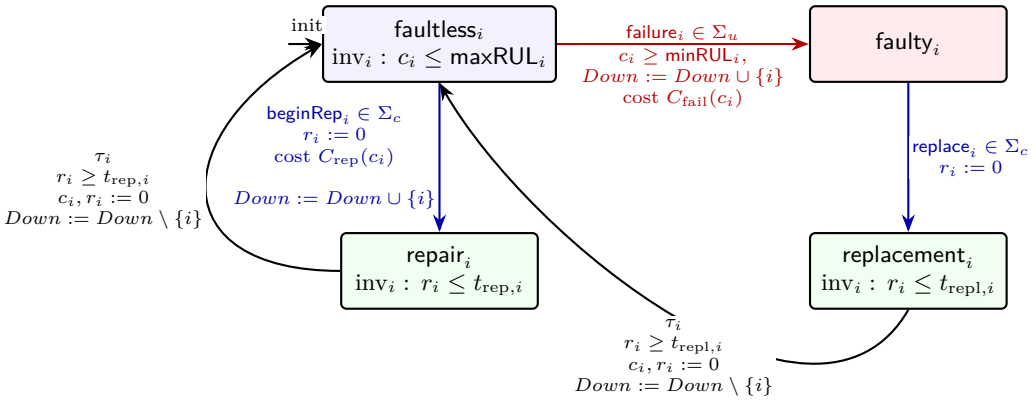

Figure 2 Dynamic Priced Timed Game Automaton $\mathcal{G}_i$ for component i. Red denotes the uncontrollable failure transition, while blue denotes controllable maintenance actions. Clock c_i measures degradation and clock r_i measures the duration of the ongoing maintenance activity. Returning to faultless$_i$ resets both clocks and samples a new hidden failure deadline in the RUL interval.

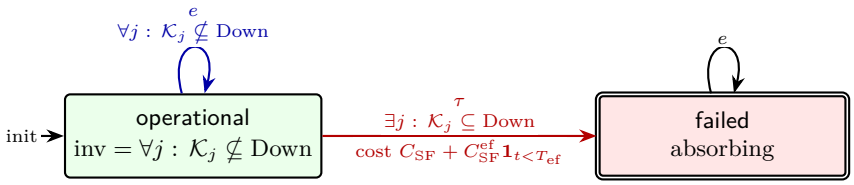

Figure 3 DPTGA of the SHM $\mathcal{M}$. After each component event e, the SHM reads the shared variable $Down$. It remains in operational as long as no minimal cut set $\mathcal{K}_j$ is entirely contained in $Down$. As soon as $\exists j : \mathcal{K}_j \subseteq Down$, the invariant on operational is violated and the autonomous τ-transition fires, moving the SHM to the absorbing location failed and incurring the system-failure cost.

5 Threshold Policy and Cross-Entropy Optimization

The objective is to find a policy that minimizes the expected cumulative operational cost over a finite horizon T while preventing catastrophic system failures. The global system is the composition $\mathcal{S} = \mathcal{G}_1 \parallel \ldots \parallel \mathcal{G}_n \parallel \mathcal{M}$ of the component automata and the SHM of Figure 3. The control problem is intrinsically a Partially Observable Markov Decision Process (POMDP): the full state at time t includes the location $\ell_{i,t}$ and both clock valuations $(c_{i,t}, r_{i,t})$ of each component automaton $\mathcal{G}_i$, but the per-component failure deadline sampled in $[\text{minRUL}_i, \text{maxRUL}_i]$ is hidden from the agent. The observable state is therefore the projection

$$o_t = \left(\ell_{i,t},\ \frac{c_{i,t}}{\text{maxRUL}_i} \right)_{i=1}^{n} \in \mathcal{O}, \tag{5}$$

where the normalized degradation ratio $c_{i,t}/\text{maxRUL}_i \in [0,1]$ indicates the extent to which the component i has aged relative to its worst-case lifespan. At each step, the agent selects an action from

$$\mathcal{A} = \{\text{wait}\} \cup \{\text{beginRep}_i, \text{replace}_i \mid i = 1, \ldots, n\}, \tag{6}$$

consistent with the controllable action sets Σ_{ci} defined in Section 4.1. The objective is to minimize the expected total cost

$$J(\pi) = \mathbb{E}_\pi[C_{\text{total}}] = \mathbb{E}_\pi\left[\sum_{\text{transitions}} f_i(t_k, v_{C,k}) + C_{\text{SF}}\,\mathbf{1}_{\mathcal{M}\to\mathsf{failed}}\right], \tag{7}$$

where $f_i(t_k, v_{C,k})$ are the DPTGA transition costs of Section 4.1 and C_{SF} is the system-failure penalty if $\mathcal{M}$ reaches the absorbing failed location (Section 4.2).

Standard value-based RL – learning the action-value function $Q(o, a)$ via Bellman updates – fails to produce useful policies in this setting. Even with optimistic initialization, the agent collapses to the degenerate "never trigger preventive maintenance" strategy. The failure is structural: the catastrophic failure penalty is a sparse terminal reward, while the cost of every preventive action is paid immediately. When the credit for avoiding a future system failure must be propagated backward through hundreds of single-step updates, the discounted contribution of the terminal penalty becomes negligible relative to the immediate maintenance cost, and the greedy policy learns to avoid all interventions. This pathology is well-known for value-based RL with sparse, long-horizon rewards and factored credit assignment.

To circumvent this difficulty, we restrict the policy to a compact parameterized class and optimize it directly at the episode [2] level using the CEM, which evaluates each candidate policy by its total cost over complete simulation episodes and is therefore immune to the credit-assignment problem. Here, although the learned maintenance policy may exhibit a simpler structure, this parameterization enables a direct and transparent explanation of the optimization results. We restrict the agent to a compact, human-interpretable policy class parameterized by a vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n) \in [0, 1.5]^n$. Given observation $o_t = (\ell_{i,t}, c_{i,t}/\mathsf{maxRUL}_i)_{i=1}^n$, the action for component i is:

$$\pi_{\boldsymbol{\theta}}(o_t)_i = \begin{cases} \mathsf{replace}_i & \text{if } \ell_{i,t} = \mathsf{faulty}_i, \\ \mathsf{beginRep}_i & \text{if } \ell_{i,t} = \mathsf{faultless}_i,\ \frac{c_{i,t}}{\mathsf{maxRUL}_i} \geq \theta_i, \\ & \quad \text{and } \forall \mathcal{K}_j \ni i:\ (\mathcal{K}_j \setminus \{i\}) \not\subseteq \mathit{Down}_t, \\ \mathsf{wait} & \text{otherwise,} \end{cases} \tag{8}$$

where the third condition on the $\mathsf{beginRep}_i$ branch is the *subsystem-safe* predicate: it blocks any preventive action that would complete a minimal cut set and immediately trigger a system failure. Formally, it requires that, for every cut set $\mathcal{K}_j$ containing i, at least one component of $\mathcal{K}_j$ other than i is still available (not in Down_t). The policy has only n continuous parameters, one per component, interpretable as an age-fraction threshold: component i is sent for preventive repair as soon as its normalized degradation $c_{i,t}/\mathsf{maxRUL}_i$ reaches θ_i, provided the action is architecturally safe. By construction, the policy provably never triggers a preventive action that would by itself cause a system failure.

The search domain is clipped to $[0, 1.5]^n$ rather than $[0, 1]^n$. Any $\theta_i > 1$ encodes the "never trigger preventive maintenance on component i" sub-policy: the invariant $c_{i,t} \leq \mathsf{maxRUL}_i$ on location $\mathsf{faultless}_i$ (Figure 2) prevents $c_{i,t}/\mathsf{maxRUL}_i$ from ever exceeding 1 while $\mathcal{G}_i$ remains in $\mathsf{faultless}_i$, so the $\mathsf{beginRep}_i$ branch is never reached. Allowing values above 1 lets CEM discover that forgoing preventive maintenance is sub-optimal for components in tight cut sets, while it may be cost-efficient for components protected by abundant k-out-of-n redundancy.

[2] An episode is a simulation of $\mathcal{S}$ over the fixed horizon T, terminating either at $t = T$ or when $\mathcal{M}$ enters failed, whichever comes first.

Algorithm 1 CEM optimization of the threshold policy (8).

Data: DPTGA simulator $\mathcal{S}$; population size N_p; elite fraction ρ; number of generations Gen; episodes per candidate K; $\boldsymbol{\mu}_0$, $\boldsymbol{\sigma}_0$
Result: Optimal threshold vector $\boldsymbol{\theta}^\star$.

Initialise $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}_0$, $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}_0$;
for $gen = 1, \ldots, Gen$ **do**
  Sample $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(N_p)} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ and clip each entry to $[0, 1.5]$;
  for $k = 1, \ldots, N_p$ **do**
    $J^{(k)} \leftarrow \frac{1}{K} \sum_{e=1}^{K} C_{\mathrm{total}}\big(\pi_{\boldsymbol{\theta}^{(k)}}; \omega_{gen,k,e}\big)$; `// avg. cost over` K `episodes with different RUL draws` $\omega_{gen,k,e}$
  $Id \leftarrow$ indices of the $\lceil \rho N_p \rceil$ candidates with lowest $J^{(k)}$;
  $\boldsymbol{\mu} \leftarrow \frac{1}{|Id|} \sum_{k \in Id} \boldsymbol{\theta}^{(k)}$;
  $\boldsymbol{\sigma} \leftarrow \mathrm{std}\Big(\{\boldsymbol{\theta}^{(k)}\}_{k \in Id}\Big) + \varepsilon$;
return $\boldsymbol{\theta}^\star \leftarrow \boldsymbol{\mu}$;

Because π_θ has only n parameters and the cost landscape is noisy but smooth, we optimize θ with the CEM (Algorithm 1). At each generation gen, CEM samples N_p candidate vectors from a diagonal Gaussian $\mathcal{N}(\mu_g, \mathrm{diag}(\sigma_g^2))$ and evaluates each candidate by its average total cost $J(\theta^{(k)})$ over K simulated DPTGA episodes. Each episode is identified by a random seed $\omega_{g,k,e}$ that encodes all stochastic draws of the simulation – in particular the hidden failure deadlines sampled uniformly in $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$ for each component i – so that averaging $C_{\mathrm{total}}\big(\pi_{\theta^{(k)}}; \omega_{g,k,e}\big)$ over $e = 1, \ldots, K$ yields a Monte-Carlo estimate of the expected cost defined in Eq. (7). CEM then retains the $\lceil \rho N_p \rceil$ elite candidates with the lowest cost and refits the Gaussian to these elites. Since the cost is evaluated at the episode level, CEM bypasses the credit-assignment difficulty that defeats step-level value-based RL.

Regarding the scalability of the approach, the temporal complexity of CEM is in $N_p \times Gen \times K$. Note that, as it is a simulation-based algorithm, the simulator $\mathcal{S}$ does not require the explicit computation of the composition $\mathcal{G}_1 \parallel \ldots \parallel \mathcal{G}_n \parallel \mathcal{M}$ but only simulates some runs from it. It means that the space complexity of the algorithm is linear with the number of components n. The quality of the result is then based on a proper selection of N_p and Gen that depends on the simulation space defined by $\mathcal{S}$.

6 Case Study and Results

6.1 Experimental Setup

We consider a multi-component system comprising seven distinct components, denoted A through G. The functional architecture of the system is shown in Figure 4. The system fails as soon as either subsystem 1 (A AND B down) or subsystem 2 (any 4 of 5 components in $\{C, D, E, F, G\}$ down) is realized. These failure conditions induce the system minima cut sets which can be extracted with several classical algorithms as explained in Section 3.2. The component parameters used in this case study (RUL bounds, maintenance durations and cost coefficients) are reported in Table 1.

The CEM-trained policy is benchmarked against three traditional strategies. *Corrective* maintenance triggers a replacement only after a component has failed. *Periodic* maintenance triggers a preventive intervention round-robin over the components every 60 time units

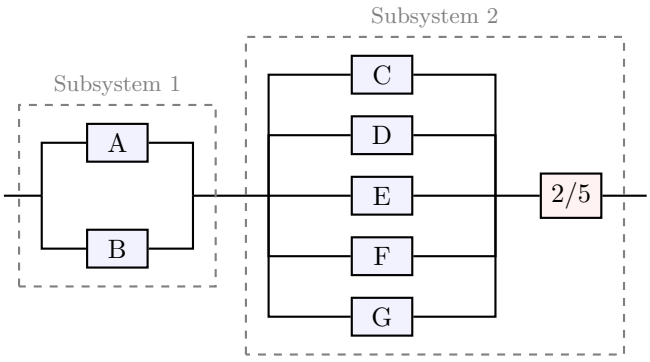

Figure 4 Functional architecture of the studied system.

Table 1 Component parameters.

ID	minRUL	maxRUL	C_{comp}	t_{rep}	t_{repl}	$C_{\text{rep\_rate}}$	$C_{\text{rep\_fix}}$	$C_{\text{repl\_fix}}$
A	120	200	200	10	15	30	50	100
B	130	170	180	5	8	20	40	80
C	120	250	190	8	12	25	45	90
D	100	180	110	6	10	15	30	60
E	150	220	120	7	11	20	35	70
F	130	200	120	7	11	20	35	70
G	90	140	90	6	10	15	30	60

(that is an empirically chosen interval intended to reflect the cost impact of systematic maintenance), regardless of degradation level. *Condition-based* maintenance is a threshold heuristic that triggers a preventive intervention as soon as $c_i \geq \mathsf{minRUL}_i$.

The time horizon is $T = 1200$, the early-failure threshold $T_{\text{ef}} = 1000$, the base failure penalty 10 000 with an additional 10 000 when the system fails before T_{ef}. The CEM agent is trained with $Gen = 20$ generations, a population of $N_p = 20$, an evaluation budget of $K = 10$ episodes per candidate, and elite fraction $\rho = 0.25$. Initial parameters are $\boldsymbol{\mu}_0 = 0.7\,\mathbf{1}_n$ and $\boldsymbol{\sigma}_0 = 0.25\,\mathbf{1}_n$. Training takes $\sim$72 s on a single CPU core, including 4 000 stochastic episodes. The four strategies are evaluated on a fresh batch of 800 independent episodes sharing the same seed offset to remove between-policy variance (2 000 episodes for the final reported numbers). Costs are reported as means, standard deviations, medians and 95$^{\text{th}}$ percentiles; survival is the fraction of episodes in which the system remained operational over the entire horizon.

6.2 Performance Comparison

Table 2 reports the headline metrics. Three observations stand out. First, the corrective and periodic baselines incur both a higher mean cost and a much lower survival rate ($\approx$40%): under the calibration of Section 4 the catastrophic failure penalty dominates the cost budget once the system fails before the horizon, and neither corrective nor a 60-step periodic schedule provides enough preventive throughput to keep the seven components inside their joint safety envelope. Second, the condition-based policy, in the current setting, is the safe risk-averse strategy: it achieves 100% survival at a deterministic cumulative cost of 16, 135, because triggering preventive maintenance at $c_i < \mathsf{minRUL}_i$ guarantees that no component ever

Table 2 Comparison of maintenance strategies (2 000 evaluation episodes; horizon $T = 1200$).

Strategy	Mean cost	Std	Median	P95	Surv. (%)
Corrective	24 176.3	10 583.3	30 593.2	38 002.1	38.3
Periodic	26 625.4	10 745.5	32 497.2	40 612.9	42.4
Condition-based	16 134.8	0.0	16 134.8	16 134.8	**100.0**
RL (CEM)	**13 793.2**	2 476.0	**13 537.1**	14 104.7	98.7

reaches its failure window. It should be noted, however, that this guarantee breaks down when all components belonging to a minimal cut share the same minRUL value: in that specific configuration (that we avoid intentionally), their preventive actions are scheduled simultaneously, the system crosses a failure cut before any intervention can be achieved, and the survival rate of this policy collapses to 0%.

Third, the CEM-trained threshold policy attains a mean cost of 13 793 and a *median* cost of 13 537 at 98.7% survival, that is, a 15% improvement of the mean cost and a 16% improvement of the median cost over the condition-based heuristic. The improvement stems from the agent exploiting the architectural redundancies of the system: in subsystem 2 where two of five components must remain operational, the agent entirely forgoes preventive maintenance on component F (whose learned threshold exceeds 1, see Section 6.3) while keeping tight schedules for the components that belong to smaller, more dangerous cut sets. Crucially, the residual failure rate is only 1.3%, and the 95^{th}-percentile cost of 14 105 is now close to the median, confirming that the policy essentially eliminates the heavy-tail risk that would be present in more aggressive parameterizations. The cost–survival trade-off is therefore *Pareto*: condition-based is the conservative pole, RL the cost-efficient pole, and the practitioner can pick one or the other depending on how the failure penalty is weighted by the operating context. Indeed, for safety-critical applications, a practitioner might increase the failure penalty (C_{SF}) to steer the CEM policy toward 100% survival. However, within the current policy class, this parameter heavily penalizes failure but cannot strictly guarantee survival. A practical solution is a hybrid strategy where the condition-based heuristic acts as a strict safety fallback: the CEM policy optimizes costs when degradation is low, and the heuristic overrides it once degradation approaches critical thresholds to ensure survival. Figure 5 shows the corresponding cost distributions, and Figure 6 shows the cost–survival trade-off across the four strategies.

Figure 6 summarises the cost–survival trade-off: the CEM policy achieves the lowest expected cost among the evaluated strategies, whereas the condition-based policy provides the conservative point with perfect survival. The convergence plot of Figure 7 confirms that CEM reaches its plateau within twenty generations, well within the training budget reported above.

6.3 Policy Interpretation and Validation

The learned threshold vector is

$$\boldsymbol{\theta}^{\star} \approx (0.60,\ 0.75,\ 0.88,\ 0.81,\ 0.81,\ 1.03,\ 0.88), \tag{9}$$

which translates into the per-component preventive triggers reported in Table 3. Three observations:

- Component F receives the highest threshold ($\theta^{\star}_F = 1.03 > 1$), which lies above the domain ceiling: the invariant $c_F \leq \mathsf{maxRUL}_F$ forces $\mathcal{A}_F$ out of the faultless location before c_F/maxRUL_F can reach 1.03, so preventive maintenance on F is *never* triggered.

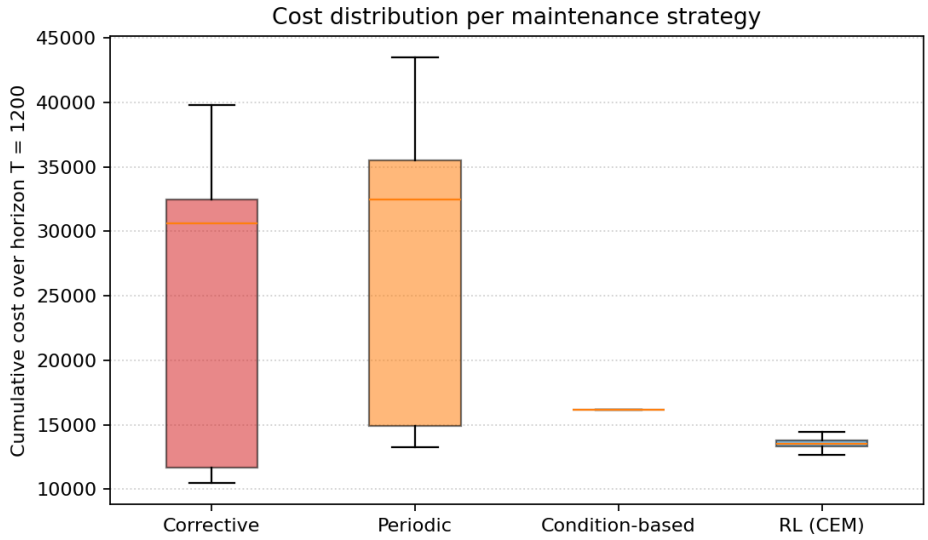

Figure 5 Cumulative-cost summary over the horizon $T = 1200$. Markers show the mean and median costs reported in Table 2; vertical intervals show the empirical standard deviation and the 95^{th} percentile.

Table 3 Learned per-component preventive triggers ($\text{age-trigger}_i = \theta_i^\star \cdot \text{maxRUL}_i$).

Component	$\theta_i^\star$	maxRUL_i	age-trigger_i	minRUL_i
A	0.60	200	120	120
B	0.75	170	128	130
C	0.88	250	220	120
D	0.81	180	146	100
E	0.81	220	178	150
F	1.03	200	>200	130
G	0.88	140	123	90

Because F belongs only to the larger four-component cuts of subsystem 2, the k-out-of-n redundancy makes proactive intervention on F unnecessary: allowing it to fail correctively is cheaper than the maintenance overhead it would incur.

- The remaining components (C, D, E, G) receive thresholds in the range 0.81–0.88, all above their respective $\text{minRUL}_i/\text{maxRUL}_i$ ratios. The policy therefore consistently delays preventive maintenance beyond the local condition-based trigger whenever the subsystem retains sufficient spare redundancy.

The policy is *architecture-aware*: it does not apply the same margin to all components. It anticipates maintenance on the tight $\{A, B\}$ cut set, entirely skips preventive maintenance on F thanks to the k-out-of-n redundancy, and allows the remaining components in subsystem 2 to age beyond the local condition-based trigger when the subsystem still has sufficient spare capacity.

To rule out an implementation artefact, the simulator was instrumented to log every preventive trigger fired by the CEM policy across 50 independent episodes (a total of 60 000 time steps). The *subsystem-safe* predicate of Eq. (8) was satisfied at every single firing: the

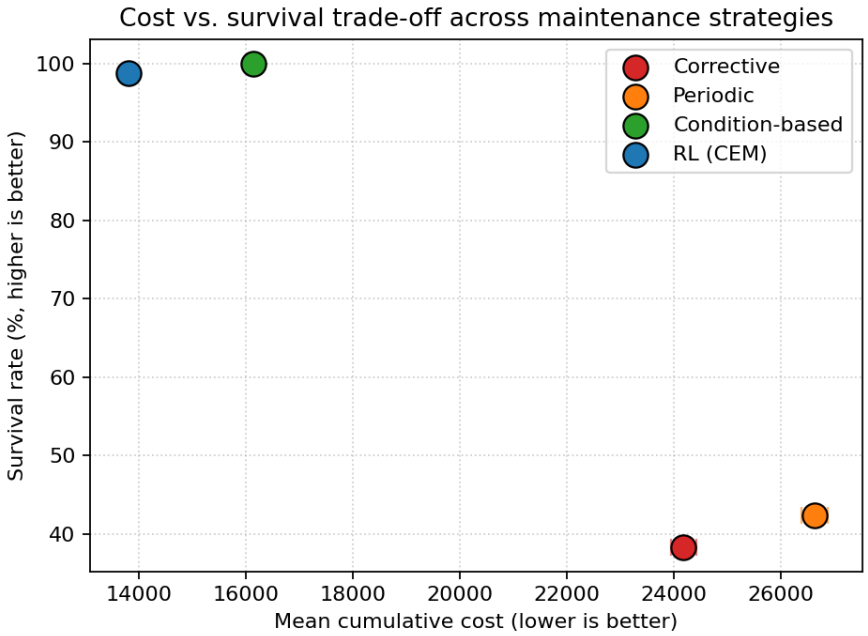

Figure 6 Cost–survival trade-off across the four strategies. The CEM policy strictly dominates the corrective and periodic baselines, and forms a Pareto frontier with the condition-based heuristic.

agent never triggered a preventive maintenance that would itself bring the system below the failure boundary. The 1.3% residual failure rate is therefore due solely to the stochastic RUL sampling – adverse draws in which two components of the same minimal cut set fail before either of them has reached its CEM threshold – not to a violation of the designed safety constraint.

The deterministic cost incurred by the condition-based policy was cross-checked against the closed-form per-component cycle cost predicted by Eq. (3). The analytical sum of $\sim 16\,135$ matches the empirical 16 135 to within 0.1%, confirming that the updated cost parameters are mutually consistent. The simulator and the cost model are therefore in agreement.
Overall, the CEM policy leverages the architectural redundancies revealed by the minimal cut sets in a way that the purely local condition-based rule cannot: components belonging to the tightest cuts trigger preventive maintenance earlier (or before their failure window opens, as with component B), while components shared by many large cuts in the k-out-of-n block are either allowed to age further before any intervention or, in the extreme case of component F, left to fail correctively because the subsystem redundancy makes proactive maintenance economically sub-optimal.

7 Conclusion and perspectives

We presented a full pipeline that synthesizes a predictive maintenance policy for a multi-component system from its functional architecture. Components and the System Health Monitor are modeled as a network of priced timed game automata, and the model semantics is reproduced verbatim by a lightweight explicit-state simulator that bypasses the state-space explosion of a model-checking back-end. On a seven-component case study, this enables a population-based policy search (CEM) over a compact, interpretable threshold policy with

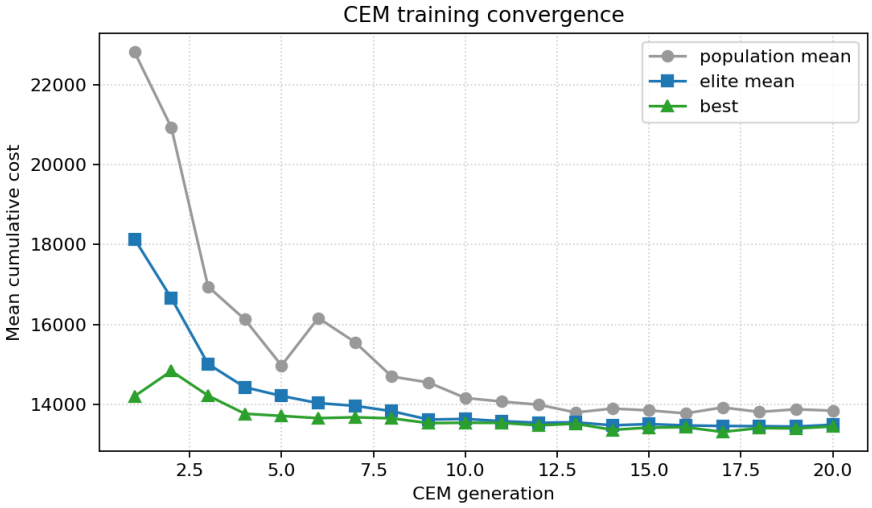

Figure 7 Cross-Entropy Method training curve. The mean of the elite population converges within twenty generations, confirming that the seven-dimensional threshold space is well-conditioned for population-based search.

one parameter per component. The learned policy reaches a mean cumulative cost of 13 793 and a median of 13 537 at a 98.7% survival rate, improving on the best rule-based strategy by 15% in mean cost and 16% in median cost, and strictly dominating the corrective and periodic baselines on both cost and reliability. The 95^{th}-percentile cost of 14 105 confirms that the policy nearly eliminates the heavy-tail failure risk. A key interpretive finding is that the agent learns to forgo preventive maintenance entirely on component F, relying instead on the k-out-of-n redundancy of subsystem 2, while maintaining tight schedules for the more critical cut-set members. The condition-based heuristic and the CEM policy form a Pareto frontier in the cost–survival plane, leaving the practitioner the freedom to pick the operating point that best fits the failure-cost ratio of the asset under management.

This work is preliminary and opens several directions for future research. i) The current cost model assumes that the residual value of a component decreases linearly with its degradation clock and that every maintenance action restores the component to an as-good-as-new state. Both assumptions are standard but debatable. On the one hand, alternative economic depreciation models (declining-balance, sum-of-years-digits) could better capture the non-linear loss of asset value in practice. On the other hand, the hypothesis of perfect maintenance could be relaxed, for instance, by adopting the Kijima model [13], in which a repair of type I reduces the effective age of the component by a fraction of the elapsed degradation, or a repair of type II resets the age of the entire system. Embedding a Kijima-type imperfect maintenance model into the DPTGA transitions would make the cost structure richer and the learned policy more realistic. ii) The DPTGA network introduced in Section 4 has not yet been studied from a formal verification standpoint. Questions of decidability (is the optimal maintenance strategy computable?) and complexity (what is the state-space growth with the number of components and clocks?) remain open and the statistical evidence reported in Section 6 should be complemented. iii) In the present work, the RUL intervals $[\mathsf{minRUL}_i, \mathsf{maxRUL}_i]$ are fixed parameters shared across all episodes. Real PHM pipelines, however, produce predictions whose uncertainty varies over time: the interval narrows as the component ages and the prognostic model accumulates observations. Given

the computational efficiency of the CEM approach, the optimization could simply be re-run at each prognostic update. Alternatively, assuming successive prognostic predictions remain relatively stable, the existing policy could be efficiently fine-tuned rather than recalculated from scratch. Another extension is to train a deep reinforcement learning agent that receives the current RUL interval as part of its state, so that the policy can adapt online to the instantaneous quality of the predictions. This would replace the static threshold class of Section 5 with a neural controller capable of exploiting time-varying prognostic confidence, at the cost of interpretability.

References

1 Mads Kronborg Agesen, Søren Enevoldsen, Thibaut Le Guilly, Anders Mariegaard, Petur Olsen, and Arne Skou. *Energy Consumption Forecast of Photo-Voltaic Comfort Cooling Using UPPAAL Stratego*, pages 603–622. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-63121-9_30.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

3 Charalampos P. Andriotis and Kostas G. Papakonstantinou. Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. *Reliability Engineering & System Safety*, 212:107551, 2021. doi:10.1016/j.ress.2021.107551.

4 Vepa Atamuradov, Kamal Medjaher, Pierre Dersin, Benjamin Lamoureux, and Noureddine Zerhouni. Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation. *International Journal of Prognostics and Health Management*, 8(3):1–31, 2017. doi:10.36001/ijphm.2017.v8i3.2667.

5 Alexandre David, Peter Gjøl Jensen, Kim G. Larsen, Marius Mikuvccionis, and Jakob H. Taankvist. Uppaal Stratego. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9035 of *LNCS*, pages 206–211. Springer, 2015. doi:10.1007/978-3-662-46681-0_16.

6 Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikuvccionis, and Danny B. Poulsen. Uppaal SMC tutorial. *Int. J. Software Tools for Technology Transfer*, 17(4):397–415, 2015. doi:10.1007/s10009-014-0361-y.

7 Phuc Do, Van-Thai Nguyen, Alexandre Voisin, Benoit Iung, and Waldomiro Alves Ferreira Neto. Multi-agent deep reinforcement learning-based maintenance optimization for multi-dependent component systems. *Expert Syst. Appl.*, 245(C), July 2024. doi:10.1016/j.eswa.2024.123144.

8 JB Fussell, EB Henry, and NH Marshall. Mocus: A computer program to obtain minimal sets from fault trees. Technical report, Aerojet Nuclear Co., Idaho Falls, ID (United States), 1974. URL: https://digital.library.unt.edu/ark:/67531/metadc1016681.

9 Martijn A. Goorden, Peter Gjøl Jensen, Kim G. Larsen, Andrey Samusev, Jiri Srba, and Penut Vongmasa. STOMPC: Stochastic model-predictive control with Uppaal Stratego. In *Automated Technology for Verification and Analysis (ATVA)*, volume 13505 of *LNCS*. Springer, 2022. doi:10.1007/978-3-031-19992-9_21.

10 Iwona Grobelna, Krystian Gajewski, and Andrei Karatkevich. A systematic review on the applications of uppaal. *Sensors*, 25(11):3484, 2025. doi:10.3390/s25113484.

11 Zhiyi Huang and Xiao Li. A critical review of operations research on the operation and maintenance of railway systems. *Journal of Railway Science and Technology*, 1(2):47–58, 2025. doi:10.1016/j.jrst.2025.08.001.

12 Manfred Jaeger and Kim Guldstrand Larsen. Reinforcement learning for discretized euclidean mdps. In Bernhard Steffen, editor, *Bridging the Gap Between AI and Reality*, pages 312–335, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-75434-0_22.

13 Masaaki Kijima. Some results for repairable systems with general repair. *Journal of Applied Probability*, 26(1):89–102, 1989. doi:10.2307/3214319.

14 Rajesh Kumar, Bhavesh Narra, Rohan Kela, and Siddhant Singh. Afmt: Maintaining the safety-security of industrial control systems. *Computers in Industry*, 136:103584, 2022. doi:10.1016/j.compind.2021.103584.

15 Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 9268 of *LNCS*, pages 156–171. Springer, 2015. doi:10.1007/978-3-319-22975-1_11.

16 Melih Cemal Kushan and Seyid Fehmi Diltemiz. Aircraft health monitoring systems (ahms) applications in aviation. In Melih Cemal Kuşhan and Seyid Fehmi Diltemiz, editors, *Aircraft Manufacturing, Safety and Control*, chapter 5. IntechOpen, London, 2025. doi:10.5772/intechopen.1013417.

17 Kim G. Larsen, Marius Mikuvccionis, Marco Muniz, Jiri Srba, and Jakob H. Taankvist. Online and compositional learning of controllers with application to floor heating. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9636 of *LNCS*, pages 244–259. Springer, 2016. doi:10.1007/978-3-662-49674-9_14.

18 Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. Fault tree analysis, methods, and applications a review. *IEEE transactions on reliability*, 34(3):194–203, 2009. doi:10.1109/TR.1985.5222114.

19 Yu Liu, Yuanjian Chen, and Tao Jiang. A deep reinforcement learning approach for maintenance planning of multi-component systems with complex structure. *Neural Computing and Applications*, 35(26):19625–19642, 2023. doi:10.1007/s00521-023-08542-9.

20 Alberto Pliego Marugán. Applications of reinforcement learning for maintenance of engineering systems: a review. *Advances in Engineering Software*, 183, 2023. doi:10.1016/j.advengsoft.2023.103487.

21 Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. doi:10.1038/nature14236.

22 Lars Mönch, John W. Fowler, and Scott J. Mason. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*. Springer, 2013. doi:10.1007/978-1-4614-4472-5.

23 Saeed Najafi and Chi-Guhn Lee. A deep reinforcement learning approach for repair-based maintenance of multi-unit systems using proportional hazards model. *Reliability Engineering & System Safety*, 234:109179, 2023. doi:10.1016/j.ress.2023.109179.

24 Oluwaseyi Ogunfowora and Homayoun Najjaran. Reinforcement and deep reinforcement learning-based solutions for machine maintenance planning, scheduling policies, and optimization. *Journal of Manufacturing Systems*, 70:244–263, 2023. doi:10.1016/j.jmsy.2023.07.014.

25 Luca Pinciroli, Piero Baraldi, Guido Ballabio, Michele Compare, and Enrico Zio. Optimization of the operation and maintenance of renewable energy systems by deep reinforcement learning. *Renewable Energy*, 183:752–763, 2022. doi:10.1016/j.renene.2021.11.052.

26 Luca Pinciroli, Piero Baraldi, Michele Compare, and Enrico Zio. Deep reinforcement learning based on proximal policy optimization for the maintenance of a wind farm with multiple crews. *Energies*, 14(20):6743, 2021. doi:10.3390/en14206743.

27 Antoine Rauzy. New algorithms for fault trees analysis. *Reliability Engineering & System Safety*, 40(3):203–211, 1993. doi:10.1016/0951-8320(93)90060-C.

28 Alberto Regattieri, A Giazzi, Mauro Gamberi, and Rita Gamberini. An innovative method to optimize the maintenance policies in an aircraft: General framework and case study. *Journal of air transport management*, 44:8–20, 2015. doi:10.1016/j.jairtraman.2015.02.001.

29 Pauline Ribot, Yannick Pencolé, and Michel Combacau. Generic characterization of diagnosis and prognosis for complex heterogeneous systems. *International Journal of Prognostics and Health Management*, 4(2), 2013. doi:10.36001/ijphm.2013.v4i2.2120.

30 Michael Rubin and Dongping Du. A review of dynamic fault tree analysis and capacity degradation for complex redundant systems. *Quality and Reliability Engineering International*, 41(3):1161–1181, 2025. doi:10.1002/qre.3693.

31 Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004. doi:10.1007/978-1-4757-4321-0.

32 Enno Ruijters, Dennis Guck, Peter van Noort, and Mariëlle Stoelinga. Modelling smart buildings using fault maintenance trees. In *Computer Performance Engineering (EPEW)*, volume 11178 of *LNCS*, pages 110–125. Springer, 2018. doi:10.1007/978-3-030-02227-3_8.

33 Enno Ruijters, Daniel Reijsbergen, Pieter-Tjerk de Boer, and Mariëlle Stoelinga. Better railway engineering through statistical model checking. In *Int. Symp. on Leveraging Applications of Formal Methods (ISoLA)*, volume 9952 of *LNCS*, pages 151–165. Springer, 2016. doi:10.1007/978-3-319-47166-2_10.

34 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017. doi:10.48550/arXiv.1707.06347.

35 Ferhat Tamssaouet, Khanh Tp Nguyen, Kamal Medjaher, and Marcos Eduardo Orchard. System-level failure prognostics: Literature review and main challenges. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 237(3):524–545, 2023. doi:10.1177/1748006X221118448.

36 Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992. doi:10.1007/BF00992698.

37 Enrico Zio. Prognostics and health management (phm): Where are we and where do we (need to) go in theory and practice. *Reliability Engineering & System Safety*, 218:108119, 2022. doi:10.1016/j.ress.2021.108119.