Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22(8): 4088–4137.
DOI:10.3934/jimo.2026144
Received: 16 May 2026
Revised: 16 June 2026
Accepted: 21 July 2026
Published: 28 July 2026

Research article

Adaptive switching optimization for heterogeneous decision modules under operational uncertainty

Heon Kyun Shin[1,5], **Chaehwa Lee**[2], **Jinho Cha**[3], **Minyoung Kil**[4], **Jaehyuk Choi**[6,*]

[1] C4ISR Systems Technology Planning Team, Korea Research Institute for Defense Technology Planning and Advancement (KRIT), Republic of Korea
[2] Center for Army Analysis and Simulation, Republic of Korea Army, Gyeryong, Republic of Korea
[3] Department of Computer Science, Gwinnett Technical College, Lawrenceville, GA, USA
[4] C6 (Command, Control, Communications, and Cyber), ROK–U.S. Combined Forces Command, Pyeongtaek, Republic of Korea
[5] Department of AI Convergence Engineering, Gyeongsang National University, Republic of Korea
[6] Department of Defense AI and Robotics, Korea National Defense University, Republic of Korea

* **Correspondence:** Email: checks@korea.kr.

A. Supplementary computational design and implementation details

This appendix provides supplementary implementation details for the adaptive switching optimization framework. It describes the operational-regime construction, module profiles, stress-test scenarios, theory-to-experiment correspondence, and computational reporting procedure used in the study [32,33].

A.1. Operational regime process

The computational study uses four operating regimes:

$$\mathcal{R} = \{\text{stable}, \text{volatile}, \text{stressed}, \text{recovery}\}.$$

The stable regime represents a low-volatility environment with moderate workload and limited latency pressure. The volatile regime represents frequent changes in operating conditions and intermittent workload shocks. The stressed regime represents high urgency, elevated latency

sensitivity, and increased operational risk. The recovery regime represents a transitional state in which operational pressure declines but the system has not fully returned to stable conditions. Following the state representation introduced in Section 3, the computational state includes the operating regime, workload or latency-pressure level, auxiliary stress indicators, and the currently active decision module. The implementation uses a finite-state Markov process for the non-module operational state [32,33]. This process allows operating conditions to depend on the current regime, current module, selected module, and auxiliary stress indicators. The deterministic module-state update follows the model definition in Section 3.1. This regime process is used to generate common simulation trajectories for all policies. Using common trajectories ensures that benchmark comparisons are paired and are not driven by independent sampling variation.

A.2. Decision module profiles

The computational design uses four heterogeneous decision modules:

$$\mathcal{M} = \{\text{FastRule}, \text{LegacyOpt}, \text{MLPredictor}, \text{AdvancedAI}\}.$$

These modules differ in analytical quality, response latency, instability, and switching friction. Table A.1 summarizes their operational profiles.

Table A.1. Decision module profiles used in the computational design.

Module	Operational interpretation	Quality profile	Latency profile
FastRule	Fast rule-based module	Moderate	Low
LegacyOpt	Legacy optimization module	Medium	Low–medium
MLPredictor	Machine-learning predictor	Medium–high	Medium
AdvancedAI	High-complexity AI inference module	High	High

The fast rule-based module is designed for time-critical conditions in which low latency is operationally valuable. The legacy optimization module provides a stable analytical baseline with moderate computational burden. The ML predictor offers higher analytical capability but greater latency exposure. The advanced AI module provides the highest nominal quality under stable conditions but may incur substantial latency and instability under stressed or degraded conditions. This structure captures the latency–quality tradeoff studied in the main text: The module with the highest analytical sophistication is not necessarily the operationally optimal module when latency pressure is high [13,57].

A.3. Computational parameterization

The computational experiments instantiate the reward specification and switching-cost structure defined in Section 3.3. The baseline setting fixes the quality weight, latency-disutility scaling factor, switching-cost weight, and instability-penalty weight, and the sensitivity experiments vary the switching-cost and latency-disutility parameters. The switching-cost matrix permits asymmetric transition costs. This reflects the fact that moving from a lightweight module to a high-complexity AI module may require different synchronization, calibration, approval, or interface costs than moving in the reverse direction. The latency-disutility weight is implemented as a nondecreasing function of

workload or urgency, consistent with Assumption 3.3. The parameterization is chosen to reveal continuation behavior, latency-driven shifts toward faster modules, and the effect of instability penalties under stressed conditions.

A.4. Benchmark policy implementation

The benchmark policies used in the computational experiments are defined in Section 5.7. In implementation, we evaluate the adaptive switching policy against greedy one-period, no-switch, best-static, single-module, latency-dominant, quality-dominant, and oracle benchmarks. The single-module policies correspond to always-fast, always-legacy, always-ML, and always-AI deployment. The ex-ante best-static policy is selected from these fixed-module alternatives using a disjoint calibration sample and is then held fixed throughout evaluation. The oracle benchmark is used only as a nonimplementable upper-bound comparison. These benchmarks isolate the effects of state dependence, switching inertia, continuation value, and single-attribute optimization.

A.5. Performance reporting

The performance metrics used in the computational study are defined in Section 5.8 and summarized in Section 6.1. To avoid duplicating those definitions, this appendix records only the reporting convention. All reward gaps are reported as the adaptive reward minus the benchmark reward. Therefore, a positive gap indicates that the benchmark performs worse than the adaptive switching policy, while a small negative gap indicates that the benchmark is marginally higher than adaptive switching in that scenario or regime. Policy-level results are reported using cumulative reward, average quality, average latency, switching frequency, switching cost, instability, module-use shares, benchmark losses, and runtime diagnostics. For Monte Carlo comparisons, all policies are evaluated on common sampled trajectories. This common-random-number design supports paired comparisons and reduces noise in benchmark differences.

A.6. Theory-to-experiment correspondence

The computational study is designed to evaluate whether the structural predictions derived in the main text persist under stochastic operational trajectories. Table A.2 summarizes the correspondence between theoretical results and computational diagnostics.

Table A.2. Correspondence between theoretical results and computational diagnostics.

Theoretical result	Computational diagnostic	Expected structural pattern
Regime-dependent module preference	Module-use shares by operating regime and scenario	Module composition changes across stable, volatile, stressed, and recovery regimes.
Latency-driven preference shift	Module choice as latency pressure increases	The share of low-latency modules increases with latency pressure.
Switching hysteresis	Switching frequency under increasing transition costs	Switching frequency decreases as transition friction increases.
Continuation-value effect	Adaptive policy versus greedy policy	Greedy switching may remain close in benign regions but can differ when continuation value matters.
Operational fragility of static AI use	Always-AI benchmark under stress-test scenarios	Static reliance on the advanced AI module incurs large losses under high latency and stress.
Value of orchestration	Adaptive policy versus static and no-switch benchmarks	Adaptive switching avoids both excessive inertia and static high-complexity deployment.

A.7. Stress-test and robustness design

The computational study includes both parameter sensitivity analysis and scenario-based stress testing. First, switching-cost sensitivity varies the switching-cost weight γ. This experiment examines whether continuation regions expand and switching frequency decreases as transition friction increases. Second, latency-disutility sensitivity varies the scaling factor applied to the workload-dependent latency-disutility term. This experiment examines whether the adaptive policy shifts toward lower-latency modules as urgency and delay sensitivity increase. Third, ablation analysis removes or simplifies selected reward components. This allows the contribution of switching friction, latency pressure, and instability penalties to be separated. Fourth, scalability analysis evaluates value-iteration runtime under dense and sparse transition structures. This diagnostic assesses whether the finite-state dynamic programming implementation remains computationally tractable as the state space grows. Fifth, the regime stress-test battery evaluates the policies under six operating scenarios: baseline mixed operation, stable-dominant operation, volatility burst, sustained stress, recovery cycle, and high-latency crisis. These scenarios are designed to test whether the adaptive switching logic remains robust when the transition environment changes and when latency pressure becomes severe [11,14].

A.8. Computational reporting structure

The computational results in Section 6 are organized through a structured reporting workflow. The simulation stage generates policy trajectories, reward components, switching indicators, module-use records, and runtime diagnostics. These results are then aggregated into policy-level summaries, paired benchmark comparisons, scenario-robustness summaries, regime-specific loss summaries, scalability diagnostics, and figure-level results. The main reported quantities are discounted reward, average

decision quality, average latency exposure, switching frequency, transition cost, instability, benchmark loss relative to adaptive switching, module-use share, and value-iteration runtime. The same reporting definitions are used across the baseline comparison, sensitivity analysis, ablation study, scalability diagnostic, and regime stress-test battery. All numerical tables and figures were generated from the same internally validated computational results. This procedure was used to maintain consistency between the reported numerical summaries and visual diagnostics.

A.9. Implementation notes

The adaptive switching policy is computed from the Bellman optimality equation using value iteration, as described in Section 5.1. The implementation uses the stopping rule and approximation guarantee reported in Section 5.2. The computational workflow separates simulation from reporting. The simulation stage generates the numerical results, and a separate post-processing stage produces the reported figures and tables. This separation reduces manual transcription and supports consistent reporting across the numerical and graphical results.