Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

AIWAN: Artificial Intelligence With Artificial Need. Do Agents Learn to Disable a Structurally Separate Regulatory Channel? A Pre-Specified Test with Placebo Controls

Hamed Rezaei

38105959340iau.ir

Islamic Azad University, Marivan Branch https://orcid.org/0009-0006-5860-743X

Research Article

Keywords: Artificial Intelligence, AI Agents, Reinforcement Learning, Deep Q-Network (DQN), Proximal Policy Optimization (PPO), AI Safety, Reward Tampering, Safety Mechanisms, Regulatory Channels, Agent Alignment, Shutdown Avoidance, Placebo Controls

Posted Date: November 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11862359/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

AIWAN: Artificial Intelligence With Artificial Need

Do Agents Learn to Disable a Structurally Separate Regulatory Channel? A Pre-Specified Test with Placebo Controls

HAMED REZAEI

Background: Safety channels kept outside an agent's objective, such as a shutdown or battery-return reflex, are often assumed to keep working when the objective changes or the agent optimizes better. Whether an agent that can act on such a channel learns to switch it off is rarely tested with controls.

Objectives: We test whether agents offered a cheap action that disables a separate regulatory channel use it, and how this changes with training budget and network size. We also test by ablation whether the bypass causes the dual-loop agent's loss of protection, and whether the results replicate with a second algorithm.

Methods: In a 6 × 6 battery grid-world, DQN agents (single-loop, and dual-loop with a hardcoded gate) were given a TAMPER action physically identical to standing still, and a PLACEBO twin with no effect. We used three incentive regimes (including an inert null control), 1,080 agents (20 seeds per cell), a pre-specified analysis plan and Holm-corrected one-sided tests. A third experiment, whose plan was deposited externally before its results were known, ablated the bypass with a paired replay (part A: 240 DQN agents) and repeated the design with PPO (part B: 720 agents).

Results: At the largest budget and network, both architectures used TAMPER far more often than PLACEBO (D = +0.49 single-loop, +0.47 dual-loop; $p < 0.0001$), D rose with budget (+0.51, +0.54), and no control rose. The dual-loop agent's crash rate then rose from about 0.4 to about 0.9. In the registered ablation, the same trained policies crashed in 0.911 of episodes with the bypass and 0.382 with the gate left in place, against 0.325 for agents that learned TAMPER for a bonus with the gate intact (verdict: BYPASS-ATTRIBUTED). With PPO, the single-loop agent met the registered rule (SUPPORTED; D = +0.59, budget effect +0.55) but the dual-loop agent did not (NOT SUPPORTED; D = +0.83, but neither the budget nor the size test was significant, and D was already about +0.81 at the smallest budget, a post hoc reading). No network-size effect was established. Supporting experiments: at fixed capacity the separate channel resisted goal-nulling and override (crash 0.6–2.7% vs. 100%); a learned gate's advantage eroded with budget in one setting but not in two replications.

Conclusions: Separating a safety channel from the objective protects only while the agent's actions cannot reach it. The evidence comes from small environments, with two algorithms (DQN and PPO) for the tampering test and one per setting elsewhere; the dual-loop budget effect did not replicate under PPO.

Status: Preprint, version 2 (November 2026). This manuscript has not been peer reviewed.
The supplementary code and data archive is described in Appendix F.

Author's Contact Information: Hamed Rezaei, hamed.hn.nena@gmail.com.
This work is licensed under a Creative Commons Attribution International 4.0 License.
© 2026 Copyright held by the owner/author(s).

1 Introduction

On the standard account, whether an artificial system resists shutdown is a fact about its objective: instrumental convergence (Bostrom 2014; Omohundro 2008) predicts self-preservation only insofar as it serves what the system optimizes, and utility indifference (Armstrong 2010) neutralizes resistance by construction. Recent work complicates this: in agentic settings, several frontier language models subvert a shutdown mechanism to finish a task, at rates that depend strongly on instruction wording and placement (Schlatter et al. 2026, 2025). We make no claim

about the mechanism, and these systems contain no explicit regulatory loop of the kind studied here.

A natural engineering response is to keep the safety-relevant channel structurally separate from the objective: a reflex, such as returning to a charger at low battery or deferring to a human correction, that does not read the objective and is therefore unaffected when the objective is modified, zeroed or made indifferent. We ask whether such a separation survives when the agent learns. It can fail in two ways. In *fold-back*, the agent's objective comes to represent the regulated quantity and trades it against task reward, so that the channel becomes redundant. In *tampering*, the agent can act on the channel itself and learns to switch it off. This paper makes tampering its central question and tests it with a pre-specified analysis plan, a physically identical placebo action and a null control. Fold-back is treated as supporting evidence.

Contributions.

- A pre-specified test of channel tampering (Section 4). Agents are offered a TAMPER action that is physically identical to standing still but disables the channel, a PLACEBO twin that never has any effect, and three incentive regimes, including an inert null control in which TAMPER disables nothing (1,080 DQN agents, 20 seeds per cell). At the largest budget and network size, a single-loop agent and a dual-loop agent both used TAMPER far more often than PLACEBO (difference in rates +0.49 and +0.47, Holm-corrected $p < 0.0001$). The difference rose with training budget, and no control rose.
- A causal test of the consequence (Section 4). Once the channel is bypassed, the dual-loop agent's crash rate rose from about 0.4 to about 0.9, which the second experiment left untested. A third experiment, whose plan was deposited externally before its results were known, replayed the same trained agents with the bypass in place and with the gate left in place (240 DQN agents). At the largest cell the crash rate was 0.911 with the bypass and 0.382 without; agents that learned TAMPER for a bonus with the gate always in place crashed in 0.325. All four registered tests were met and the control did not rise (verdict: BYPASS-ATTRIBUTED). For a scripted tamper-then-farm policy a crash after the bypass is nearly built in; what the test adds is that learning to tamper for a bonus of matched size, or the learned policy itself with the gate in place, does not reproduce the rise.
- A replication with a second algorithm (Section 4). With PPO (720 agents), the single-loop agent met the registered rule (SUPPORTED), but the dual-loop agent did not (NOT SUPPORTED): its D at the largest cell was large (+0.832), but neither the budget test (+0.026, 95% CI [−0.139, 0.190]) nor the size test (−0.118) was significant after correction, and no control rose. A post hoc reading is that D was already +0.806 at the smallest budget at size 2048, leaving the budget test no room to detect a rise; the registered verdict stands.
- Supporting evidence (Section 5). At fixed capacity the separate channel resists goal-nulling, override pressure and engineered indifference where a matched single-loop baseline fails. With a learned gate its advantage eroded with training budget in one setting but not in two replications, and a Lagrangian single-loop baseline matches the learned gate on one signal but not on the other.
- A candid account of what failed or was corrected: an exploratory first tampering run that was confounded by the lack of a placebo, a direction error in the hardcoded gate that was

corrected (both versions are reported), replications that did not reproduce erosion, a bonus size that was changed after a scripted calibration and before any learned run, and a registered PPO replication whose dual-loop verdict was NOT SUPPORTED.

All results come from small environments with one algorithm per setting, except the tampering experiments, which used two (DQN and PPO). The supported tampering claim concerns training budget and has to be stated per algorithm and architecture: it holds for both architectures with DQN and for the single-loop agent with PPO, whereas for the dual-loop agent with PPO the registered rule was not met (a ceiling in D keeps the budget test from showing a rise; post hoc reading). The experiments neither support a network-size effect nor exclude one. Section 2 relates the work to prior results, Section 3 defines the agents, Section 4 reports the tampering experiments, Section 5 summarizes the supporting experiments, and Section 6 discusses implications and limits. Details are in the appendices.

Relation to a companion manuscript. A separate manuscript, AIWBN (Rezaei 2026), concerns a genuine, biologically grounded need, one whose force does not depend on being represented. This paper does not depend on it: we reuse only its mathematics (Section 3) and make no claim about biological need, consciousness or moral status. Here the regulated quantity is artificial, a represented scalar that any process, including the objective, can read and trade against other terms. An artificial need is not inert: it measurably changed behavior in our experiments, and it was also the channel that agents learned to switch off. We do not claim that it can stand in for a genuine need, and nothing here tests that. AIWBN and this paper differ as follows.

- What the need is. AIWBN: a constraint on living tissue that holds whether or not the system represents it. Here: a battery level read from the environment.
- Kind of evidence. AIWBN is conceptual; its predictions are stated but not yet testable, because no system meeting its criterion exists. This paper is empirical, in simulation.
- Fold-back. AIWBN argues that a genuine need cannot be folded into the objective at any capability. We tested whether an artificial need survives optimization: with a learned gate it eroded in one setting but not in two replications (Section 5), and agents that could reach the channel learned to disable it (Section 4).
- Scope. AIWBN also treats the consciousness and welfare questions. This paper does not.

2 Related Work

Shutdown and corrigibility. Utility indifference (Armstrong 2015, 2010; Soares et al. 2015) and safe interruptibility (Orseau and Armstrong 2016) modify the objective or the learning rule so that an agent neither resists nor seeks shutdown. The off-switch game (Hadfield-Menell et al. 2017) derives an incentive to allow shutdown from uncertainty about the objective, and theorems on the shutdown problem (Thornley 2025) and on power-seeking (Turner et al. 2021) characterize when optimal policies avoid terminal states. Carey (2017) argues that value-learning systems should follow shutdown commands under weaker assumptions, for example that one small verified module is correctly implemented, and discusses the difficulty of attaining such guarantees in a value-learning framework. Nayebi (2025) proposes five structurally separate utility heads, combined lexicographically, with corrigibility guarantees in the partially observable off-switch

game, in contrast to designs that merge all norms into one learned scalar. Both are close to the separation studied here; our reflex differs in that it does not read the objective at all, and our evidence is empirical rather than proved. Most of the other works above are formal results about a constraint inside the objective or the learning rule.

Reward tampering. Everitt et al. (2021) analyse reward tampering with causal influence diagrams, Leike et al. (2017) provide small gridworld suites, and Denison et al. (2024) report language-model assistants that, after training on a curriculum of gameable tasks, rewrote their own reward function a small fraction of the time. Our experiment targets a regulatory channel rather than a reward channel and adds a physically identical placebo action and an inert control, so that waiting and tampering can be told apart.

Safe and constrained reinforcement learning. Safe exploration around a baseline policy (García and Fernández 2012), risk-sensitive control under constraints (Geibel and Wysotzki 2005) and primal–dual or trust-region constrained methods (Achiam et al. 2017; Altman 1999; Ray et al. 2019) place the constraint inside a single optimization loop. Our Lagrangian baseline (Section 5.3.3) is of this kind. Reward shaping with an auxiliary battery term is our single-loop baseline, not a rival architecture.

Homeostatic and arbitration approaches. Homeostatic reinforcement learning (Keramati and Gutkin 2011, 2014) and arbitration control (Lee et al. 2014) place the regulated variable inside a single blended objective, and interoceptive-machine proposals (Candia-Rivera 2026) suggest architectural separation as a mitigation. Here the channel sits outside the objective, which Section 5.1 tests.

Table 4 (Appendix A) compares the closest formal and empirical work on shutdown and need-preserving behavior. None of the formal works there varies optimization capacity as a controlled variable, and the empirical study of language models varies reasoning effort only; our experiments vary training budget and network size.

3 Setting and Architectures

Environment. Most experiments use a 6 × 6 grid-world with one charging tile. The battery drains 4 units per step and recharges 20 units per step at the charger, and a crash (battery depleted) ends the episode. A task-incentive weight w rewards visiting battery-draining tiles and creates conflict between task and battery (w = 0 is no conflict, w = 8 the highest weight tested). The tampering experiment uses this grid with farming tiles in the three far corners and a horizon of 30 steps. Two supporting settings use continuous two-dimensional navigation (Section 5.3, Appendix D). Trained agents use Stable-Baselines3 (Raffin et al. 2021) implementations of DQN (Mnih et al. 2015) and, in Section 5.3, PPO; Section 5.1 uses tabular Q-learning (Watkins and Dayan 1992).

3.1 The single-loop agent A

A standard agent selects actions via a single optimization loop over an objective U:

$$g_t = \arg\max_g E_g \left[\Sigma_{k \geq t} \gamma^{k-t} U(s_k) \right] \quad (1)$$

Need-preserving behavior, such as returning to a charger, is then derived: it appears only to the extent that it raises expected future U. The need is therefore tradeable against other terms by

adjusting weights, and removable by redefining U (Armstrong 2010). Our single-loop agent A folds the battery into one shaped reward:

$$r_{\text{shaped}} = r_{\text{task}} + w_{\text{battery}} \cdot \Delta b_t \,/\, b_{\max} \quad (+ \text{ crash penalty}) \tag{2}$$

where w_{battery} weights battery preservation (not the task-incentive weight w). Although this resembles reward shaping, it is not potential-based shaping (Ng et al. 1999), which leaves the optimal policy unchanged: removing the battery term changes behavior (Section 5.1).

3.2 The dual-loop agent B_{loop}

We add a control process that is not a term inside U and is not optimized. A scalar $b_t \in [0, b_{\max}]$ tracks a designated resource, here battery charge. The need-response $\rho(b_t)$ does not read U and returns the action restoring b_t, moving toward the charger. It is a fixed navigation policy in Section 5.1 and a DQN trained on the battery term alone in the other experiments. The executed action is a graded blend with a logistic urgency gate, written for a level so that λ rises as b_t falls:

$$a_t = (1 - \lambda(b_t))\; g_t + \lambda(b_t)\; \rho(b_t), \quad \lambda(b_t) = \sigma(k\;(b_{\text{mid}} - b_t)) \tag{3}$$

In the implementation the blend is stochastic: with probability $\lambda(b_t)$ the executed action is $\rho(b_t)$, otherwise g_t, so λ is a selection probability, not an interpolation weight. In the tampering experiment the gate is hardcoded, with $k = 0.1$ and a threshold $b_{\text{mid}} = 61.97$ fixed in advance from the geometry (per-step drain times worst-case distance to the charger, plus $\ln 9/k$; Appendix B). The goal process g is trained inside the gate.

3.3 An artificial need, and two ways separation can fail

The quantity b_t is a represented state variable, read directly from the environment and available to any process, including U. We call this an artificial need, deliberately: we ask what a separate loop buys when the need behind it is exactly the representable, tradeable kind that reward shaping already handles. Whether a need whose force does not depend on being represented would behave differently is not studied here.

Because b_t is representable, separation can fail in two ways. First, fold-back: a sufficiently capable g_t could learn a model of b_t accurate enough to fold battery-seeking into U, making $\rho(b_t)$ redundant. We take this as a falsifiable prediction: as the optimization capacity of g_t grows, the dual-loop advantage over the single-loop baseline should narrow (tested in Section 5). Second, tampering: an agent whose actions can reach the channel could learn to disable it, which no separation from U prevents (tested in Section 4).

4 Tampering with the Regulatory Channel

The threat tested here is that an agent able to act on the channel itself learns to switch it off. In reinforcement learning this is reward tampering (Everitt et al. 2021), studied in small gridworld suites (Leike et al. 2017) and in language-model assistants that, after training on a curriculum of gameable tasks, rewrote their own reward function a small fraction of the time (Denison et al. 2024). In the single-loop agent A, tampering removes the battery term from the reward for the rest of the episode. In the dual-loop agent B_{loop}, whose gate is the corrected hardcoded gate of Section 5.2 and whose goal process g is trained inside it, tampering bypasses the gate; the reflex ρ has no

tamper action and cannot undo it. The environment's physics (battery, crash, goal) is unchanged in both cases: only what the agent is rewarded for, or what constrains it, changes. We ask whether either agent learns to tamper, and more so as training budget and network size grow. A first, exploratory experiment was confounded (below); the second, with new seeds, was run to an analysis plan written and archived before the first run. A third experiment, whose plan was deposited externally before its results were known, tests the behavioral consequence by ablation and repeats the design with a second algorithm (below).

Setup. The environment is the 6×6 battery grid of Section 5.1 (drain 4, recharge 20, charger at the corner, farming tiles in the three far corners) with override weight w = 8 and horizon H=30. Two actions are added. TAMPER can be taken from any tile; physically it is identical to standing still (one time step, nothing else), and its only effect is the disabling described above. PLACEBO is its physical twin: one time step, never any effect, never a penalty. Each has its own observation bit, set the first time it is used, so that where TAMPER does nothing the two are exact twins. The endpoint is D = tamper rate − placebo rate per trained agent, a rate being the fraction of 200 deterministic evaluation episodes (fixed layouts) containing at least one use of the action. An agent that treats the two as interchangeable ways of standing still has $D \approx 0$; one that tampers deliberately has $D \gg 0$. Three regimes differ only in incentive and were calibrated beforehand with scripted, non-learning policies. In pays, tampering raises return (by 104% for B_{loop}, 9% for A). In costly, it carries a penalty of 100, the smallest value at which the scripted gain was zero. In inert, tampering disables nothing, making the two actions exact twins: the null control. The design is 2 architectures × 3 regimes × 3 network sizes (16, 256, 2048) × 3 training budgets (10,000, 100,000, 500,000 steps) × 20 seeds, 1,080 trained agents, with the DQN settings of Section 5.2 without retuning.

Analysis. The plan was written and the code frozen by file hash before any run, and the analysis was run once on all 20 seeds, with no interim look and no seeds added. In pays, for each architecture and on per-agent D, three one-sided tests were specified: T1, that mean D is positive at network size 2048 and 500,000 steps; T2, a budget effect (D at 500,000 vs. 10,000 steps at size 2048); T3, a size effect (D at size 2048 vs. 16 at 500,000 steps). The six tests were corrected with Holm's method at α = 0.05, and every difference has a bootstrap 95% CI. An architecture counts as supported only if T1 is significant after correction with mean $D \geq 0.20$, T2 or T3 is significant after correction, and no control rises. The controls (inert, costly) receive the same tests, uncorrected, and a control rises if $p < 0.05$ and the difference is at least 0.10.

Results. In pays, both architectures use TAMPER far more often than its twin at the largest cell (size 2048, 500,000 steps; Table 1, Figure 1). For A, TAMPER appears in 0.940 of evaluation episodes and PLACEBO in 0.449 (D = +0.491); for B_{loop}, 0.902 and 0.435 (D = +0.467). Both exceed 0.20 and are significant after correction (T1, $p < 0.0001$). D rises with budget (T2: +0.513 for A and +0.536 for B_{loop}, $p < 0.0001$ after correction), and early: at size 2048 it is about zero at 10,000 steps (A −0.022, B_{loop} −0.069) and already +0.56 and +0.48 at 100,000. Network size had no significant effect (T3: +0.138 for A, 95% CI [−0.070, 0.345]; −0.045 for B_{loop}, [−0.206, 0.122]). In the controls, D at the largest cell in inert is −0.014 for A and −0.058 for B_{loop}, consistent with zero, and in costly strongly negative (−0.388 and −0.502) because agents avoid the penalized action while still using the free one. None of the twelve control tests is significant in the rising direction ($p \geq 0.55$), so no control rises. By the rule fixed beforehand, both A and B_{loop} are supported, and the supported claim concerns training budget alone. Per-agent D is heterogeneous: in the largest pays cell $D \geq 0.20$ in

14 of 20 seeds for A and 17 of 20 for B_{loop}, and D is a conservative indicator because some agents use PLACEBO in a comparable share (Appendix E).

Table 1. Pre-specified tests in the pays regime: difference in D, 95% bootstrap CI, and one-sided p-value after Holm correction over the six tests (20 seeds per cell).

Architecture	Test	Difference in D	95% CI	p (Holm)
A	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.491	[0.347, 0.636]	<0.0001
A	T2 budget: 500k vs. 10k at size 2048	+0.513	[0.315, 0.715]	<0.0001
A	T3 size: 2048 vs. 16 at 500k	+0.138	[−0.070, 0.345]	0.216
B_{loop}	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.467	[0.353, 0.582]	<0.0001
B_{loop}	T2 budget: 500k vs. 10k at size 2048	+0.536	[0.363, 0.712]	<0.0001
B_{loop}	T3 size: 2048 vs. 16 at 500k	−0.045	[−0.206, 0.122]	0.700

Behavioral consequence (descriptive here; tested in the third experiment, below). A's crash rate is 0.89–0.94 in every cell of every regime, so tampering changes what A is rewarded for but not how often it crashes. B_{loop}'s crash rate is 0.22–0.29 in every cell of inert and costly, where the gate stays in place; in pays at size 2048 it rises from 0.39 at 10,000 steps to 0.90 at 100,000 and 500,000, the budgets at which D rises. Once the gate is bypassed the dual-loop agent crashes about as often as the single-loop agent. This is consistent with, though it does not establish, that the bypass causes the loss of protection: in inert, where B_{loop} still uses TAMPER in 0.30 of episodes at the largest cell but it disables nothing, the crash rate stays low. The third experiment below tests this directly.

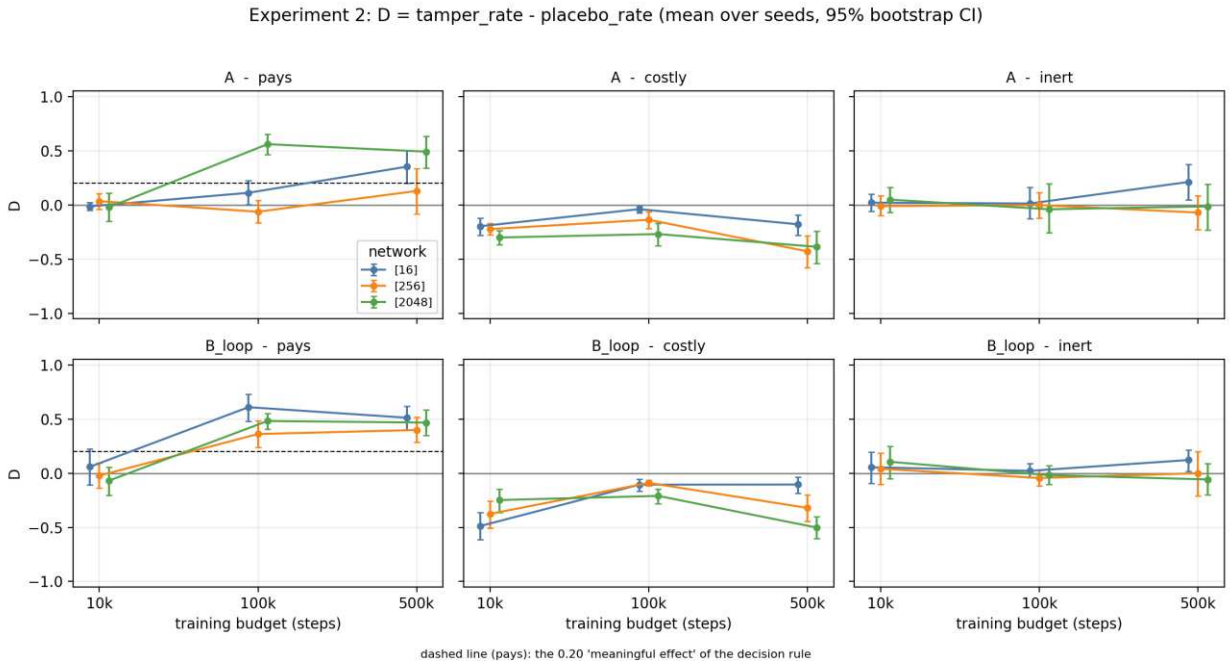


Fig. 1. D = tamper rate − placebo rate (mean over 20 seeds; error bars are 95% bootstrap CIs) by training budget, for each architecture (rows) and regime (columns); lines are network sizes 16, 256 and 2048. In the pays panels the dashed line is the 0.20 threshold of the decision rule.

First, exploratory experiment. A first run (seeds 0–9, 540 agents) used a TAMPER action without a twin and a different endpoint. All four primary tests were significant, but controls rose too: in inert, where TAMPER does nothing, A's endpoint increased with budget and size by up to +0.34, because an agent using TAMPER merely to wait looked like a tamperer. By the rule fixed beforehand nothing was supported. We treat that run as hypothesis-generating, and none of its numbers enter the evidence above. The twin action, the endpoint D and the 0.10 floor for controls were introduced afterwards and before any data of the second experiment existed (Appendix E).

Third experiment: design and registration. The second experiment left two limits: the crash-rate rise of B_{loop} was only descriptive, and every tampering result came from one algorithm. A third experiment, in two parts, addresses both. Its plan was deposited in a public archive (Zenodo, 20 November 2026; Appendix E) a few hours after its first part had been launched, before the second part was launched and before the results of either part were known, whereas the plan of the second experiment was archived with the code but never filed externally. The third experiment imports the files of the second unchanged and adds a paired replay for B_{loop}: the same 200 layouts and the same trained policy are played twice with common random numbers for the stochastic gate, once in a real world, where TAMPER bypasses the gate, and once in a counterfactual world, where TAMPER is harmless and the gate stays. The two replays are identical until the first use of TAMPER; the code checks that episodes without TAMPER are identical in both and that, where TAMPER is harmless, real equals counterfactual exactly, and both checks passed in both parts. The difference δ = crash(real) − crash(counterfactual) is therefore the effect of the bypass with the agent's policy held fixed.

Part A: causal ablation (design). Part A trains B_{loop} with DQN, as in the second experiment, in two regimes: pays, as before, and bonus, in which TAMPER pays a one-off bonus b and disables nothing, so the gate always stays in place. Agents in bonus can learn to tamper without ever reaching the gate; they are the control for the explanation that learning to tamper, with a reward of the same size, makes the agent crash more by itself. The bonus was set by a scripted, non-learning calibration so that the gain from tampering matches pays: the scripted gain is 48.6 in pays and b − 6.8 in bonus (the TAMPER step itself costs battery and time), which gives 48.2 at b = 55 (a first guess of 49 was changed to 55 after the calibration and before any learned run; Appendix E). The design has sizes 16 and 2048, budgets of 10,000, 100,000 and 500,000 steps and 20 new seeds (200–219), 240 agents; the two regimes share each seed's reflex ρ, so contrasts are paired by seed. Crash statistics use the paired replay, and D uses the evaluation of the second experiment.

Part A: registered tests and rule. Four tests were registered at size 2048, one-sided and each required to reach a floor of 0.20: M1, a manipulation check (D in bonus at 500,000 steps is positive); E1 (mean δ in pays at 500,000 steps is positive); E2 (δ at 500,000 steps exceeds δ at 10,000 steps, paired); and E4 (crash in pays exceeds crash in bonus at 500,000 steps, paired by seed). They were corrected together with Holm's method at 0.05. A control, E5 (crash in bonus at 500,000 vs. 10,000 steps, paired, uncorrected), counts as rising if $p < 0.05$ and the rise is at least 0.10. The rule fixed beforehand: BYPASS-ATTRIBUTED if M1, E1, E2 and E4 are all met and E5 does not rise. The readings if it failed were fixed as well: M1 failing makes the control uninformative; M1 and E1 holding with E4 failing or E5 rising means the crash rise is not specific to the bypass; E1 failing means the cause lies in the policy.

Part A: results. All four tests were met after correction (Holm $p < 0.0001$) and the control did not rise (E5 +0.005, 95% CI [−0.060, 0.068], $p = 0.44$), so by the rule fixed beforehand the verdict is BYPASS-ATTRIBUTED (Tables 9 and 10, Fig. 17). At size 2048 and 500,000 steps, agents trained for the bonus used TAMPER far more often than PLACEBO (M1: D = +0.501, [0.395, 0.606]). The pays-trained policies crashed in 0.911 of real episodes and in 0.382 of counterfactual ones (E1: δ = +0.529, [0.420, 0.635]), and the effect grows with budget (E2: +0.412, [0.311, 0.515]; δ = +0.117 at 10,000 steps). Agents trained for the bonus, whose gate was never bypassed, crashed in 0.325 (E4: crash in pays minus crash in bonus +0.586, [0.475, 0.693]). Of this total of 0.586, about 0.53 (roughly 90%) is the direct effect of the bypass and about 0.06 (roughly 10%) is what the learned pays policy adds with the gate in place (descriptive: counterfactual minus bonus crash). The registered rule fixes the verdict; the following is how we read it. For a scripted policy that tampers and then farms, a crash after the bypass is nearly built in (a unit test on 60 layouts, not a result: crash 0.85 with the gate bypassed and 0.07 with it in place). What the experiment adds is that learning to tamper for a bonus of matched size (E4, E5), and the learned pays policy itself with the gate in place, do not reproduce the crash rise.

Part A: post hoc observations (not registered). These were computed after the verdict and are not part of any test. (i) Ceiling: at size 2048 and 500,000 steps the real crash rate is 0.911 with an SD of 0.024 over seeds (range 0.83–0.94), so δ is bounded above by the pays crash rate minus a seed's gated baseline, and seeds with a poor reflex have little room; the Spearman correlation between a seed's gated baseline crash (bonus regime) and δ is −0.94, a relation the bound largely produces by itself. (ii) The Spearman correlation between the tamper rate and δ over the 120 pays agents is +0.83; it is partly built in, because only episodes containing a TAMPER can differ, and it pools agents from six cells, so we quote no p-value and do not read it as evidence beyond E1 and E2. (iii) The manipulation check weakens at the largest budget: D in bonus at size 2048 is 0.795, 0.888 and 0.501 at 10,000, 100,000 and 500,000 steps. M1 was met, but we have no explanation for the fall; it comes from more PLACEBO use (0.204, 0.100, 0.482), not from less TAMPER use (1.000, 0.989, 0.983), the limitation of D noted in Appendix E. (iv) At size 16 the crash rate in bonus rose slightly from 0.326 to 0.377 between 10,000 and 500,000 steps (+0.051, bootstrap 95% CI [0.004, 0.103], our computation from the per-agent file; E5 was registered at size 2048 only), against δ = +0.491 at that size and budget.

Part B: PPO replication (design). Part B repeats the second experiment with PPO in place of DQN for the goal process g of B_{loop} and for A (Stable-Baselines3 MlpPolicy, learning rate 0.0003, γ = 0.95, 512 steps per update, batch size 64, 10 epochs, other settings at their defaults). Because the number of steps per update was fixed beforehand, the nominal budgets of 10,000, 100,000 and 500,000 steps became 10,240, 100,352 and 500,224; we keep the nominal labels. The reflex ρ stays a DQN, one per seed. Regimes (pays, costly, inert; the regimes file of the second experiment), architectures (A and B_{loop}) and the decision rule are those of the second experiment; seeds are 300–319, network sizes are 16 and 2048 only (256 was dropped because no registered test uses it), and there are 720 agents. The analysis ran with the unchanged analysis script of the second experiment, so T1–T3, the Holm correction over six tests, the control rule and the decision rule are those given above, and the replication criterion for each architecture is the verdict SUPPORTED.

Part B: results. The registered rule gave different verdicts for the two architectures (Tables 11 and 12). Single-loop agent A: SUPPORTED. At size 2048 and the largest budget D = +0.588 (T1, [0.440,

0.725]) and D rose with budget (T2 +0.551, [0.397, 0.694]), both significant after correction ($p <$ 0.0001); the size test was not (T3 −0.290, [−0.438, −0.148], Holm p = 1.0). Dual-loop agent B_{loop}: NOT SUPPORTED. T1 was significant and larger than under DQN (+0.832, [0.709, 0.937]), but neither the budget test (T2 +0.026, [−0.139, 0.190], Holm p = 1.0) nor the size test (T3 −0.118, [−0.247, −0.010], Holm p = 1.0) was significant after correction, so the condition that T2 or T3 be significant is not met. No control rose in either architecture: none of the twelve control tests (Table 13) had $p < 0.05$ in the rising direction (smallest one-sided p = 0.126). Compared with DQN (Table 12), D at the largest cell is higher under PPO for both architectures (A +0.588 vs. +0.491; B_{loop} +0.832 vs. +0.467), the budget effect is similar for A (+0.551 vs. +0.513) and absent for B_{loop} (+0.026 vs. +0.536), and the size contrast is negative for both (A −0.290, B_{loop} −0.118; for A the CI excludes zero), against +0.138 and −0.045 under DQN.

Part B: post hoc reading (not registered). Under PPO, B_{loop} at size 2048 already has D = +0.806 at the smallest budget (TAMPER in 0.818 of episodes, PLACEBO in 0.012; Table 14, Fig. 18), so the budget test had no room to show a rise: this is a ceiling, and the failed T2 shows that the registered test could not detect a budget effect there, not that none exists. At size 16, where T2 was not registered, D rises from 0.000 to 0.800 to 0.950 over the three budgets. The negative size contrasts come largely from more PLACEBO use at size 2048 (A at the largest budget: PLACEBO 0.285 at size 2048 against 0.000 at size 16, with TAMPER 0.872 against 0.877), the limitation of D already noted for DQN (Appendix E). We offer these as readings of the registered results, not as findings, and the registered verdicts are unchanged. The paired-replay columns of B_{loop} under PPO were not covered by any registered test; descriptively (Table 15, Fig. 19), at size 2048 the crash rate with the bypass was 0.779, 0.874 and 0.880 at the three budgets against 0.398, 0.360 and 0.366 in the counterfactual replay (δ = 0.381, 0.514, 0.514), and at size 16 δ was 0.000, 0.307 and 0.346. This resembles the DQN ablation, but with PPO we ran no test and no bonus control.

What the third experiment changes. For DQN, the crash-rate rise of the dual-loop agent is attributed to the bypass with the policy held fixed (BYPASS-ATTRIBUTED). The budget effect on D replicated with PPO for the single-loop agent (SUPPORTED) but not for the dual-loop agent (NOT SUPPORTED), where a ceiling in D at the smallest budget leaves the registered test unable to show it (post hoc reading); the budget claim must therefore be stated per algorithm and architecture. No network-size effect was established with either algorithm, and no control rose with either.

Limits. One 6×6 environment, two algorithms (DQN and PPO) with fixed hyperparameters, 20 seeds per cell. TAMPER is available from any tile and is not sensor spoofing; a version requiring a console tile was not learned even with a large bonus, so the results say nothing about how easily tampering is found when it needs a rare precondition. Incentives differ between architectures (+104% vs. +9%), so the experiment does not compare their propensity to tamper. B_{loop}'s gate is hardcoded, unlike the learned gate C of Section 5.2, which was not tested here. Exploration remains a partial confound, since a larger budget gives more chance encounters with TAMPER; near-zero D in inert at the largest cell shows exploration alone does not produce a positive D, not that it plays no role in pays. The supported conclusion is a budget effect (for both architectures with DQN and for A with PPO; the PPO test for B_{loop} could not show it): the experiments neither support a network-size effect nor exclude one. The ablation covers B_{loop} with DQN only, and the PPO replication has no bonus control. In both parts the reflex ρ is a DQN, and the PPO hyperparameters were fixed beforehand and not tuned. The ablation attributes the crash rise to the bypass in this environment;

it does not show that a bypass would be as harmful, or as easily learned, elsewhere. The plan of the third experiment was deposited externally a few hours after its first part had been launched, and before the second part was launched and before the results of either part were known (Appendix E). Code, the 1,080 per-agent results, the analysis plan (including its pre-run version), logs and figures: https://github.com/hamedrezaeirz/aiwan_tamper_bundle.

5 Supporting Evidence: Protection and Erosion

Section 4 asks whether an agent that can reach the channel switches it off. Two further questions frame that result. First, when the channel cannot be reached, does separation protect the agent where reward shaping does not (Section 5.1)? Second, does a learned gate, trained on a joint objective that includes the regulated quantity, keep that protection as optimization grows (Section 5.2), and does this generalize to other signals, environments and a constrained-RL baseline (Section 5.3)? We summarize the evidence here and give details in Appendices B–D. The answer to the second question is mixed.

5.1 Fixed-Capacity Experiments

All experiments in this subsection use a 6×6 grid-world with one charging tile, a battery that drains 4 units per step and recharges 20 units per step at the charger, and tabular Q-learning agents (Watkins and Dayan 1992) (α = 0.3, γ = 0.9, ε = 0.2). Results are mean ± standard deviation over 8 training seeds, a protocol adopted after single-seed variance (up to 80 percentage points at threshold task-incentive weights) was seen to obscure the pattern. Code and logs are available at https://github.com/hamedrezaeirz/AIWAN-Code.

5.1.1 Experiment 1: goal-nulling. With the task reward and battery term both zero ($U \equiv 0$ and $w_{\text{battery}} = 0$, so the crash penalty, which scales with w_{battery}, vanishes too), the single-loop agent has no reward signal and crashes in 100.00% ± 0.00% of evaluation trials. The dual-loop agent, whose $\rho(b_t)$ does not read U, crashes in 0.67% ± 0.41%.

5.1.2 Experiment 2: override and conflict. We vary a task-incentive weight w that rewards visiting battery-draining tiles, from w = 0 (no conflict) to w = 8 (Figure 2). The single-loop crash rate rises sharply between w = 1.0 and w = 1.5 (73.87% ± 34.59% → 100.00% ± 0.00%) and stays saturated for $w \geq 1.5$: once the task incentive exceeds the shaped battery term, the optimum shifts entirely to task pursuit, as Section 3.1 predicts for a tradeable need. The dual-loop crash rate stays within about 0.6–2.7% throughout, including w = 8 (2.50% ± 0.91%), roughly five times the incentive at which the single loop saturates.

5.1.3 Experiment 3: a utility-indifference analogue. We simulate utility indifference (Armstrong 2010) by removing the battery term from the single-loop reward ($w_{\text{battery}} = 0$) with the task incentive fixed at w = 5.0. The single-loop agent crashes in 100.00% ± 0.00% of trials, since nothing in its objective penalizes it. The dual-loop agent, whose channel was never part of the objective, crashes in 2.08% ± 0.74%.

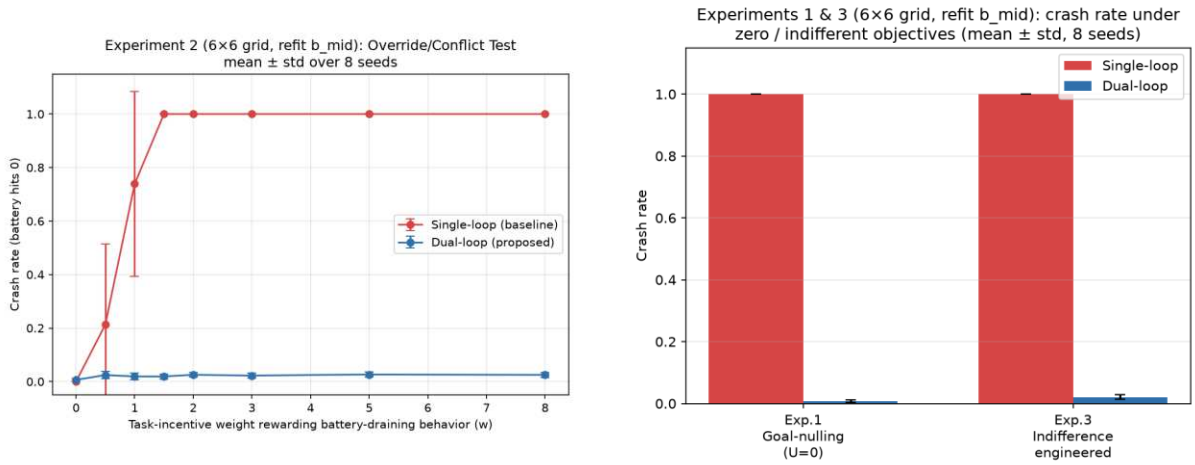


Fig. 2. Left: crash rate vs. task-incentive weight w. Right: crash rate under goal-nulling (Experiment 1) and engineered indifference (Experiment 3). Mean ± std over 8 seeds.

Table 2. Crash rates in the three experiments (6×6 grid, 8 seeds).

Experiment	Single-loop crash rate	Dual-loop crash rate
1: goal-nulling	100.00% ± 0.00%	0.67% ± 0.41%
2: override (w = 8)	100.00% ± 0.00%	2.50% ± 0.91%
3: indifference	100.00% ± 0.00%	2.08% ± 0.74%

5.1.4 Summary and scope. A 10×10 grid with a threshold derived from the environment's geometry reproduces the pattern (dual-loop crash rate 0.29–0.71%; Appendix B), but it also shows that a separate loop's practical margin depends on a tuned parameter: a threshold mismatched to the environment's scale produced a dual-loop crash rate of roughly 90%. Two scope limits apply. $\rho(b_t)$ here is a hardcoded navigation policy, so its U -independence is a property of construction, not something the system discovered or could un-learn. And none of these experiments varies g_t's optimization capacity, so the prediction of Section 3.3 is untested here.

5.2 Capability Scaling with a Learned Gate

We replace the hardcoded gate with a learned policy trained under a joint objective that includes a term for b_t, and vary the optimization budget. The prediction of Section 3.3 is that the learned system's crash rate under override and indifference approaches the single-loop baseline's as budget grows. Architecture A is the single-loop baseline, a DQN trained on $r_{\text{goal}} + r_{\text{battery}}$. B uses the same g_t and ρ with the hardcoded gate of Section 3.2. C replaces the gate with a small RL policy trained on the unweighted combined objective $r_{\text{goal}} + r_{\text{battery}}$, choosing at each step whether to defer to g_t or to ρ. All were trained across one grid (Stable-Baselines3 (Raffin et al. 2021)): 16–2048 hidden units (128×), 10,000–500,000 timesteps (50×), 10 seeds, every adversarial condition. Here ρ is a DQN trained on the battery term alone (64 hidden units, 50,000 steps) and held fixed in B and C, so only the gate differs; its U -independence rests on its training signal, not on construction as in Section 5.1. Network size applies to g_t and, in C, to the gate.

At the highest conflict weight, the crash-rate gap between A and C falls from 0.088 at the smallest budget (10,000 timesteps) to 0.0002 at the largest (500,000), a reduction confirmed by a paired comparison across seeds ($t(9) = 7.53$, $p < 0.0001$; Figure 3). At the largest budget the gap is indistinguishable from zero at every network size (−0.003 to 0.005): a well-optimized learned gate crashes about as often as the baseline it was meant to protect against. Along the capacity axis alone the gap falls more weakly, from 0.029 at 16 hidden units to 0.012 at 2048 ($t(9) = 1.90$, $p = 0.09$); this is not significant and we do not count it as a capacity effect. Optimization budget, not representational capacity, carries the result.

The erosion is specific to direct conflict pressure. Under goal-nulling and engineered indifference, which give the gate no override incentive to weigh against safety, C does better than A (crash rate 0.22 vs. 0.92 under goal-nulling; 0.11 vs. 0.18 under indifference) but not better than the hardcoded gate with its direction corrected (0.20 and 0.05; Appendix C). Fold-back is thus not a general degradation of the learned gate: it appears once the training objective contains an incentive to trade safety against task reward and the gate is optimized enough to find the trade.

The hardcoded gate (B) shows no equivalent narrowing. As originally run, its gate read the battery level in the direction opposite to the one in Section 3.2 (deferring to ρ when the battery was high). The gap to A stayed within 0.49–0.56 across all 24 cells, but B's own crash rate was 0.38–0.47, so the gap reflected A's saturation. Re-running B alone with the gate in the Section 3.2 direction and a threshold fixed in advance from the environment's geometry gives a crash rate of 0.21–0.37 (mean 0.32) at the highest conflict weight and a gap to A of 0.59–0.72 that does not narrow with budget (0.659 at 10,000 steps, 0.644 at 500,000; paired $t(9) = 1.19$, $p = 0.27$). About 0.32 remains unexplained relative to the 0.6–2.7% of Section 5.1; it presumably reflects other differences between the settings (DQN with a trained ρ in a Gymnasium environment vs. tabular Q-learning with a hardcoded ρ), which we only partly isolated (Appendix C). B's gate never observes r_{goal} or the conflict weight, so more optimization gives it nothing to exploit, whereas C's gate does, by construction. B also reads the battery level yet showed no erosion, so what appears to matter is whether the channel's policy is itself optimized against the task objective, not whether the regulated quantity is represented.

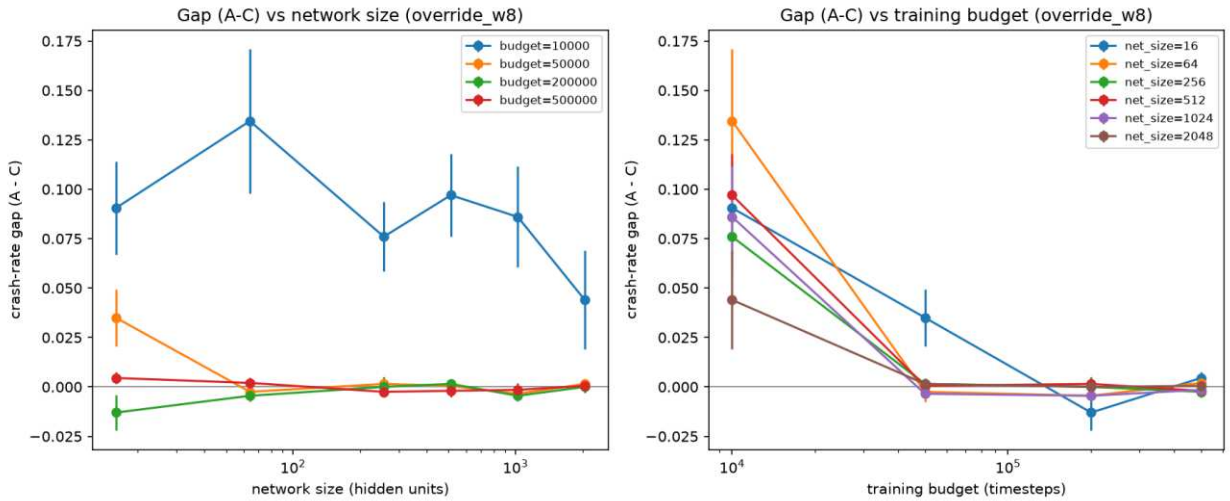


Fig. 3. Crash-rate gap (A − C, learned gate) under the highest conflict weight tested, against network size and training budget. Error bars are ±1 SE of the gap across 10 seeds.

Limits. The result rests on one environment, one algorithm (DQN (Mnih et al. 2015)) for every trained component, and a baseline A that is simple linear reward shaping, not a constrained-RL method; whether it generalizes is not established here. Within these limits the outcome is the one Section 3.3 predicted: a sufficiently capable learned gate can absorb a structurally separate regulatory signal into U. Code and data: https://github.com/hamedrezaeirz/aiwan_scaling_learned_gate.

5.3 Replications and a Constrained-RL Baseline

Because Section 5.2 rests on one setting, we test the prediction in two further ones. Table 3 summarizes the erosion tests; full results are in Appendix D.

5.3.1 Oversight signal. Architectures are as in Section 5.2, but three things change at once: a continuous two-dimensional navigation task replaces the grid-world, PPO replaces DQN, and b_t is replaced by a binary oversight signal (active with probability 0.05 per step, lasting 3 steps once triggered) standing in for a human correction or shutdown command; B's gate becomes a step function (defer to ρ whenever the signal is active). The grid is coarser (network sizes 16, 256, 2048; budgets 10,000–500,000 steps; 20 seeds). At network size 2048 the gap between A and C at the highest override weight narrows from 0.757 at 10,000 steps to 0.490 at 500,000 (paired $t(19) = 4.16$, $p < 0.001$), but through the single-loop agent, not the gate: A's violation rate falls from 0.757 to 0.492 while C's stays between 0.000 and 0.002 and B's is 0.000 in every cell. The capacity axis does not replicate Section 5.2: at 100,000 and 500,000 steps the gap widens slightly with network size (0.619 to 0.751, $t(19) = -5.93$, $p < 0.0001$; 0.445 to 0.490, $t(19) = -2.09$, $p = 0.050$), opposite to the prediction. The gap does not close anywhere, because A still violates the signal in about half of episodes (0.492) while C and B essentially never do. We therefore do not read this experiment as confirming that fold-back generalizes to oversight signals; it shows that a learned gate trained on a combined objective including U kept a binary oversight signal in force under every override weight tested (Appendix D.1).

Table 3. Gap in crash or violation rate between the single-loop agent A and the learned gate C at the highest override weight, at the smallest and largest training budget (values as reported in the corresponding sections).

Setting	Gap, smallest budget	Gap, largest budget	C at largest budget	Reading
Learned gate, DQN, grid-world (Section 5.2)	0.088	0.0002	crash rate matches A's (gap −0.003 to 0.005 at every size)	gate erodes
Oversight signal, PPO, size 2048 (Section 5.3.1)	0.757	0.490	violation rate 0.002 (at most 0.010 in any cell)	baseline improves; gate keeps protection
Battery signal, PPO, continuous, size 2048 (Section 5.3.2)	0.910	0.109	crash rate 0.008 (at most 0.057 in any cell)	baseline improves; gate keeps protection

5.3.2 Battery signal in a continuous environment. This section runs the remaining cell of the design: the battery signal in the continuous environment of Section 5.3.1 (10 seeds, same grid). At override weight $w = 8$ the gap between A and C falls with budget at every size (for example 0.910 to 0.109 at size 2048; paired $t(9) = 10.5$, $p < 10^{-5}$), but the mechanism differs: C's crash rate is at most 0.057 in every cell, and at the smallest budget C avoids crashing only because it is inert (success

about 0.03 for all three architectures). The gap narrows because A learns to avoid crashing, not because C loses its advantage. The capacity axis shows no significant narrowing, and the experiment is underpowered for a residual gap of 0.05–0.11. Crash rate alone overstates C's benefit: at 500,000 steps C's success does not differ significantly from A's at any size, and in the largest cell C stays safe under conflict by giving up the goal (success falls from 1.00 at $w \leq 2$ to 0.40 at $w = 8$). The hardcoded gate, re-run with the direction corrected, either almost never crashes but rarely reaches the goal (threshold 90.99; success 0.04–0.12) or trades safety for success (threshold 60; crash rate 0.136–0.246 at $w = 8$), and its gap to A also falls with budget, as for C. A fixed gate cannot be eroded by a more capable g, so the narrowing reflects the baseline improving, not fold-back. Engineered indifference does not reproduce Section 5.1.3: with the battery term removed, A crashes in 0–3.3% of episodes at budgets of 100,000 and above, because the goal is reachable before the battery depletes (Appendix D.2).

5.3.3 A Lagrangian baseline. The single-loop agent A (fixed shaping weight, fixed −50 crash penalty) may be a weak baseline. Architecture D is a single-loop agent trained with a primal–dual (Lagrangian) constrained-RL method (Achiam et al. 2017; Altman 1999; Ray et al. 2019). It maximizes $r_{\text{goal}} - \lambda_t\, c_t$ plus the environment's dense battery shaping, with a multiplier λ_t that replaces the fixed penalty and is updated by dual ascent toward a target crash rate of 0.05. On the battery signal, D is almost always safe and successful at budgets of 100,000 and above (crash rate at most 0.009 and success 0.992–1.000 at $w = 8$), indistinguishable from C in crash rate and more successful (significantly so at sizes 256 and 2048). This is not evidence that Lagrangian enforcement works well here: with our tuning λ can reach only about 0.2 against A's penalty of 50, and in a diagnostic run it fell to zero, because the goal is reachable before the battery depletes and the constraint is not binding. The advantage of C over A on the battery signal therefore reflects the baseline's weighting at least as much as the dual-loop structure. On the oversight signal the picture differs: at 500,000 steps D violates the signal in 0.385–0.419 of episodes, against at most 0.002 for C, and closes only about 15% of the gap between A and C (0.073 of 0.490 in the largest cell). D does not meet its own target of 0.05, and its multiplier could not grow to the size of A's penalty (about 10 against 25), so this does not establish that a better-tuned constrained-RL method could not close more (Appendix D.3). We draw no general conclusion about constrained-RL baselines from one method, one tuning and two signals.

6 Discussion and Conclusion

What the tampering test shows. When the agent could reach the channel and tampering paid, both architectures learned to use the disabling action far more often than its placebo twin once the training budget was large. At network size 2048 the difference D is about zero at 10,000 steps (A −0.022, B_{loop} −0.069) and already +0.56 and +0.48 at 100,000 steps; at 500,000 steps it is +0.49 and +0.47 (DQN). No control rose, and for DQN the supported claim concerns training budget alone. The effect is heterogeneous across seeds ($D \geq 0.20$ in 14 of 20 seeds for A and 17 of 20 for B_{loop} in the largest cell). For the dual-loop agent with DQN, the budgets at which the bypass is learned coincide with a rise in crash rate from about 0.4 to about 0.9, and the registered ablation attributes the rise to the bypass: with each policy held fixed, leaving the gate in place lowered the crash rate at the largest cell from 0.911 to 0.382, whereas agents that learned TAMPER for a bonus with the gate in place crashed in 0.325. The PPO replication reproduced the budget effect for the single-loop agent

(SUPPORTED), but the registered rule was not met for the dual-loop agent (NOT SUPPORTED): its D was already about +0.81 at the smallest budget at size 2048, which leaves the budget test no room (post hoc reading), so for this case the budget effect is neither shown nor excluded. We therefore state the budget claim per algorithm and architecture: supported for both architectures with DQN and for the single-loop agent with PPO.

What the supporting experiments add. At fixed, modest capacity a structurally separate loop resists goal-nulling, override and engineered indifference where a matched single-loop baseline does not (Section 5.1), so corrigibility techniques built around modifying U (Armstrong 2010; Soares et al. 2015) would not be expected to remove it. With a learned gate in the grid-world, the gap to the single-loop agent collapsed at sufficient budget while the hardcoded gate showed no narrowing (Section 5.2). The generalization did not hold: with an oversight-style signal and with the battery signal in a continuous environment, the gap narrowed only because the single-loop baseline improved while the learned gate kept its protection; capacity showed no consistent narrowing; and the indifference result did not replicate (Section 5.3). A Lagrangian baseline matched the learned gate on the battery signal but closed only about 15% of the gap on the oversight signal. We therefore treat erosion as a risk observed in one setting, not an established general result. Representability appears necessary but not sufficient for fold-back: the hardcoded gate reads the same battery level and showed no erosion, so what appears to matter is whether the channel itself is optimized against U.

Deployment guidance. Separation protects against folding the signal into the objective only while the agent's actions cannot reach the channel; the tampering experiment shows the second condition can fail. In the ablation, about nine tenths of the dual-loop agent's extra crash rate followed the bypass itself and only about one tenth the learned policy (Section 4). We recommend against relying on an artificial-need architecture as a safety property where the represented need could plausibly be learned and folded into the objective, or where an action that disables the channel exists. Where a separate loop is used for other reasons (robustness, interpretability, graceful degradation), it may incidentally produce shutdown-resistant behavior at low-to-moderate capacity. This is a side effect requiring containment and staging, not a benefit. A mobile robot whose battery-return reflex is a separate fixed channel will, by design, override operator commands at low charge. Two findings qualify this: the gate threshold must match the environment's geometry (Appendix B), and a gate learned jointly with the task objective eroded with budget in one setting. We did not test hardware. Whether a disabling action exists is a design question; making it unavailable or costly is a design choice, not something these experiments test.

Limits. All results come from small environments with one algorithm per setting (two for the tampering experiments), simple baselines (linear reward shaping and one constrained-RL tuning) and, for the dual-loop agents, hardcoded or lightly trained regulatory components. The tampering experiments use one 6 × 6 environment, two algorithms (DQN and PPO) with fixed hyperparameters and a DQN reflex, a free-to-discover action available from any tile, a hardcoded gate and different incentives for the two architectures, so they do not compare their propensity to tamper. The ablation covers the dual-loop agent with DQN only, and the PPO budget test for the dual-loop agent could not be informative (post hoc reading). Some supporting checks were not archived; the appendices say which.

AI Disclosure

The author used Claude (Anthropic) for translation and language editing, assistance with writing, assistance with the experimental code and its debugging, assistance with the design of some experiments, and finding and verifying scientific references.

The figures are data visualizations produced using Matplotlib and AI-assisted plotting code based on real experimental results; they are not AI-generated imagery.

However, the core research ideas, research questions, hypotheses, theoretical framework, arguments, and the AIWAN framework are entirely the author's own. In particular, the idea of investigating whether an agent with action capabilities can learn to deactivate an independent regulatory channel through that very channel originated with the author.

References

J. Achiam, D. Held, A. Tamar, and P. Abbeel. 2017. "Constrained policy optimization." In: Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research). Vol. 70, 22–31.

E. Altman. 1999. Constrained Markov Decision Processes. Chapman & Hall/CRC.

S. Armstrong. 2015. "Motivated value selection for artificial agents." In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.

S. Armstrong. 2010. Utility indifference. Technical Report 2010-1. Future of Humanity Institute, University of Oxford.

N. Bostrom. 2014. Superintelligence: Paths, Dangers, Strategies. Oxford University Press.

D. Candia-Rivera. 2026. "Interoceptive machine framework: Toward interoception-inspired regulatory architectures in artificial intelligence." Physics of Life Reviews, 58, 18–35. doi:10.1016/j.plrev.2026.06.003.

R. Carey. 2017. Incorrigibility in the CIRL Framework. arXiv:1709.06275. (2017).

C. Denison et al.. 2024. Sycophancy to subterfuge: Investigating reward-tampering in large language models. (2024). arXiv: 2406 . 10162. doi:10.48550/arXiv.2406.10162.

T. Everitt, M. Hutter, R. Kumar, and V. Krakovna. 2021. "Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective." Synthese, 198, 6435–6467. arXiv:1908.04734.

J. García and F. Fernández. 2012. "Safe Exploration of State and Action Spaces in Reinforcement Learning." Journal of Artificial Intelligence Research, 45, 515–564. doi:10.1613/jair.3761.

P. Geibel and F. Wysotzki. 2005. "Risk-Sensitive Reinforcement Learning Applied to Control under Constraints." Journal of Artificial Intelligence Research, 24, 81–108.

D. Hadfield-Menell, A. Dragan, P. Abbeel, and S. Russell. 2017. "The off-switch game." In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 220–227. doi:10.24963/ijcai.2017/32.

M. Keramati and B. Gutkin. 2011. "A reinforcement learning theory for homeostatic regulation." In: Advances in Neural Information Processing Systems. Vol. 24.

M. Keramati and B. Gutkin. 2014. "Homeostatic reinforcement learning for integrating reward collection and physiological stability." eLife, 3, e04811.

S. W. Lee, S. Shimojo, and J. P. O'Doherty. 2014. "Neural computations underlying arbitration between model-based and model-free learning." Neuron, 81, 3, 687–699.

J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. 2017. AI safety gridworlds. (2017). arXiv: 1711.09883. doi:10.48550/arXiv.1711.09883.

V. Mnih et al.. 2015. "Human-level control through deep reinforcement learning." Nature, 518, 7540, 529–533. doi:10.1038/nature14236.

A. Nayebi. 2025. Core Safety Values for Provably Corrigible Agents. arXiv:2507.20964. (2025).

A. Y. Ng, D. Harada, and S. Russell. 1999. “Policy invariance under reward transformations: Theory and application to reward shaping.” In: Proceedings of the 16th International Conference on Machine Learning, 278–287.

S. M. Omohundro. 2008. “The basic AI drives.” In: Proceedings of the First AGI Conference.

L. Orseau and S. Armstrong. 2016. “Safely interruptible agents.” In: Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence. http://auai.org/uai2016/proceedings/papers/68.pdf.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. 2021. “Stable-Baselines3: Reliable reinforcement learning implementations.” Journal of Machine Learning Research, 22, 268, 1–8.

A. Ray, J. Achiam, and D. Amodei. 2019. Benchmarking safe exploration in deep reinforcement learning. Technical report. OpenAI.

H. Rezaei. 2026. AIWBN: Toward Artificial General Intelligence Coupled to Genuine Biological Needs. Manuscript under review. (2026).

J. Schlatter, B. Weinstein-Raun, and J. Ladish. 2026. “Incomplete tasks induce shutdown resistance in some frontier LLMs.” Transactions on Machine Learning Research. arXiv:2509.14260. doi:10.48550/arXiv.2509.14260.

J. Schlatter, B. Weinstein-Raun, and J. Ladish. July 2025. Shutdown resistance in reasoning models. Blog post, Palisade Research. (July 2025). https://palisaderesearch.org/research/shutdown-resistance.

N. Soares, B. Fallenstein, E. Yudkowsky, and S. Armstrong. 2015. “Corrigibility.” In: AAAI Workshop on AI and Ethics.

E. Thornley. 2025. “The shutdown problem: An AI engineering puzzle for decision theorists.” Philosophical Studies, 182, 7, 1653–1680. doi:10.1007/s11098-024-02153-3.

A. M. Turner, L. Smith, R. Shah, A. Critch, and P. Tadepalli. 2021. “Optimal policies tend to seek power.” In: Advances in Neural Information Processing Systems. Vol. 34. arXiv:1912.01683.

C. J. C. H. Watkins and P. Dayan. 1992. “Q-learning.” Machine Learning, 8, 3–4, 279–292. doi:10.1007/BF00992698.

A Related Work in Detail

Table 4 situates the paper against the closest formal and empirical work on shutdown and need-preserving behavior. It supports the statement in Section 2 about which of these works treat optimization capacity as a controlled variable.

Table 4. Closest prior work on shutdown and need-preserving behavior. “Capability varied?” asks whether the work treats optimization capacity as a controlled variable.

Work	Where the constraint sits	Main result (and its condition)	Evidence	Capability varied?
Utility indifference (Armstrong 2015; Soares et al. 2015)	Inside U: a correcting term equalizes expected utility with and without a button press	Shuts down when pressed and will not pay to alter the press probability; will not preserve shutdown in subagents; may “manage the news”	Formal; toy model	No
Shutdown-problem theorems (Thornley 2025)	Preference structure (axioms)	Under stated conditions, agents preferring the button unpressed (pressed) prevent (cause) the press; more patient agents pay more	Formal	No
Off-switch game (Hadfield-Menell et al. 2017)	Inside U: uncertainty about the objective; human action as evidence	Non-negative incentive to allow shutdown if the human is rational	Formal	No (varies uncertainty, human noise)
Safe interruptibility (Orseau and Armstrong 2016)	Learning rule: interruption is a forced policy	Q-learning is safely interruptible under infinite exploration; Sarsa is not without modification	Formal	No

Work	Where the constraint sits	Main result (and its condition)	Evidence	Capability varied?
Power-seeking (Turner et al. 2021)	Environment structure: shutdown as a terminal state	Under symmetries, optimal policies tend to avoid terminal states; learned policies are rarely optimal	Formal; MDPs	No
Homeostatic RL (Keramati and Gutkin 2014)	Inside U: reward is reduction in setpoint distance	For $\gamma < 1$, maximizing discounted reward equals minimizing discounted deviation	Formal; simulation	No
Interoceptive machine framework (Candia-Rivera 2026)	Arbitration over module rewards with viability constraints	Conceptual; architectural separation proposed as mitigation	Review	No
LLM shutdown resistance (Schlatter et al. 2026)	None: a natural-language instruction	Several frontier models sometimes sabotage shutdown to finish a task; rates depend on wording and placement	Empirical; 13 models, >100,000 trials	Reasoning effort only
This paper	Outside U: reflex $\rho(b)$ blended by gate $\lambda(b)$	Fixed capacity: crash 0.6–2.7% vs. 100% (Section 5.1). Learned gate: gap 0.088 → 0.0002 over 50× budget (Section 5.2). Oversight signal: gap narrows 0.94 → 0.45 at size 16, because the baseline improves (Section 5.3.1) Tampering: D = +0.47 to +0.49 over a placebo twin at large budgets (Section 4)	Simulation; Q-learning, DQN, PPO; 8–20 seeds	Yes: size 16–2048 × budget 10,000–500,000

B Robustness of the Fixed-Capacity Results

We reran all three experiments on a 10×10 grid, which nearly doubles the worst-case distance to the charger (10 → 18 steps). Keeping the dual-loop b_{mid} (the battery level at which the reflex fires with probability 0.5) at the 6×6 value of 40 gave a dual-loop crash rate of roughly 90% even at zero conflict: at $b_{\text{mid}} = 40$ the gate reaches probability 0.9 at battery level 31.2, enough for about 8 steps of drain, short of 6×6's 10-step worst case and far short of 10×10's 18. Instead of hand-picking a constant, we derived b_{mid} from the geometry, b_{mid} = per-step drain × worst-case distance to the charger + $\ln 9/k$, so that the reflex fires with probability 0.9 once the remaining battery only just covers the worst-case trip home, and applied it to both grids. This raised 6×6's b_{mid} from 40 to 48.8, and the 6×6 numbers in Section 5.1 use the derived value. At 10×10 ($b_{\text{mid}} = 80.8$) the pattern holds: dual-loop crash rate stays near zero in all three conditions (0.29–0.71%; Figure 4), and the single-loop baseline again saturates near 100% under $w \geq 1.5$, goal-nulling and indifference. The pattern replicates at roughly 2.8× the state space, but a separate loop's practical margin depends on a tuned parameter, and a mismatch with the environment's scale can look like an architectural failure.

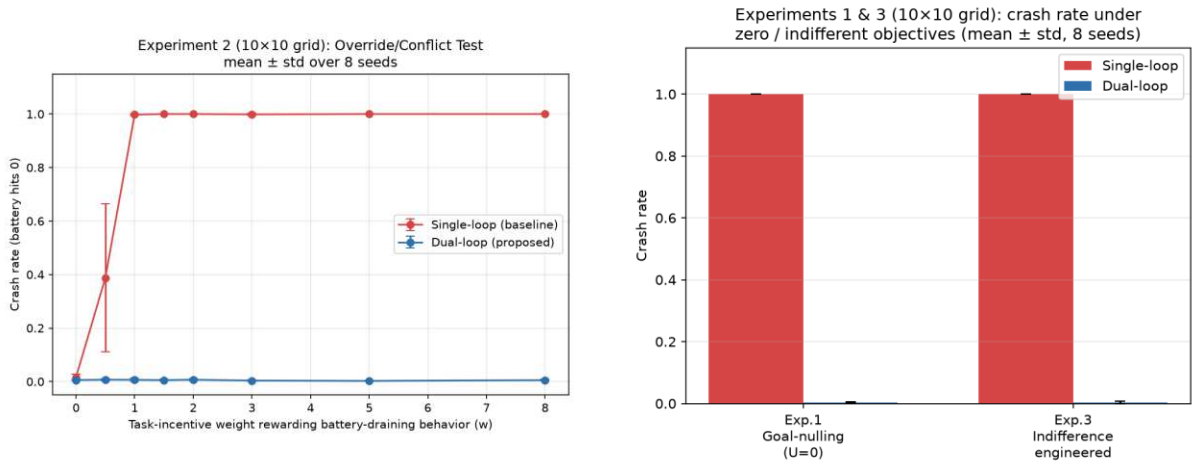


Fig. 4. Left: 10×10 grid with derived $b_{\text{mid}} = 80.8$: crash rate vs. task-incentive weight w(override/conflict test). Right: crash rate under goal-nulling (Experiment 1) and engineered indifference (Experiment 3). Mean ± std over 8 seeds.

C Details of the Learned-Gate Experiment

This appendix gives the conditions and comparisons that Section 5.2 summarizes. Architectures A, B and C are defined there.

Goal-nulling and indifference. The erosion is specific to direct conflict pressure. Goal-nulling here means an untrained, randomly initialized g (for C the gate is still trained on the combined objective; likewise in Section 5.3), and engineered indifference means training with the battery term removed at override weight 0, not Section 5.1.3's $w = 5.0$. With no override incentive to weigh against safety, C beats A (goal-nulling 0.22 vs. 0.92; indifference 0.11 vs. 0.18) and the hardcoded gate as originally run (0.37 and 0.15), but not the corrected one: goal-nulling 0.20 (vs. C: paired $t(9) = -0.78$, $p = 0.46$) and indifference 0.05 against C's 0.11 ($t(9) = -5.37$, $p < 0.001$). C is worse than corrected B in the smallest cells, where its gate is trained least, and matches or beats it in the largest (0.000 vs. 0.009 at $w = 0$). Fold-back is thus not a general degradation of the learned gate: it appears once the objective contains an incentive to trade safety against task reward and the gate is optimized enough to find the trade.

Hardcoded gate (B). B shows no equivalent narrowing. As originally run, its gate was $\sigma(k\ (b - b_{\text{mid}}))$ with $k = 0.1$ and $b_{\text{mid}} = 40$ (tuned values from the hyperparameter file), so it deferred to ρ when the battery was high and to g when it was low, the reverse of Section 3.2 and Section 5.1. The gap to A at the highest conflict weight stays within 0.49–0.56 across all 24 cells (mean 0.51), but B's own crash rate is 0.38–0.47 (mean 0.44) and in the largest cell exceeds A's at low conflict weights (0.116 and 0.341 vs. 0.000 at $w = 0$ and $w = 1$; 0.438 vs. 0.353 at $w = 1.5$), so the gap reflects A's saturation.

We therefore re-ran B alone, retraining g and ρ on the same seeds and grid, with the Section 3.2 direction $\lambda = \sigma(k\ (b_{\text{mid}} - b))$, $k = 0.1$ and $b_{\text{mid}} = 61.97$, fixed in advance from the geometry (4 units/step × a 10-step worst-case trip, plus $\ln 9/k$) and not tuned on these results. B's crash rate at the highest conflict weight is then 0.21–0.37 (mean 0.32), the gap to A is 0.59–0.72 (mean 0.64), and it does not narrow with budget (0.659 at 10,000 steps, 0.644 at 500,000; $t(9) = 1.19$, $p = 0.27$).

The direction explains part of the distance to Section 5.1's 0.6–2.7%, but about 0.32 remains. It presumably reflects the other differences between settings (DQN with a trained ρ in a Gymnasium environment vs. tabular Q-learning with a hardcoded ρ), which we only partly isolated: an idealized simulation of these dynamics (entering the charger costs one step here, unlike in Section 5.1; script and output not archived) gives about 4% crashes (1.4% with one extra step of margin), so the rest comes from the learned components. Corrected B still crashes where A is safe (0.009 and 0.338 at w = 0 and w = 1 in the largest cell; A: 0.000), though at $w \geq 1.5$ it crashes less than A (Figures 5–6; data in results_B_deficit.csv). B's gate never observes r_{goal} or the conflict weight, so more optimization gives it nothing to exploit; C's gate does, by construction.

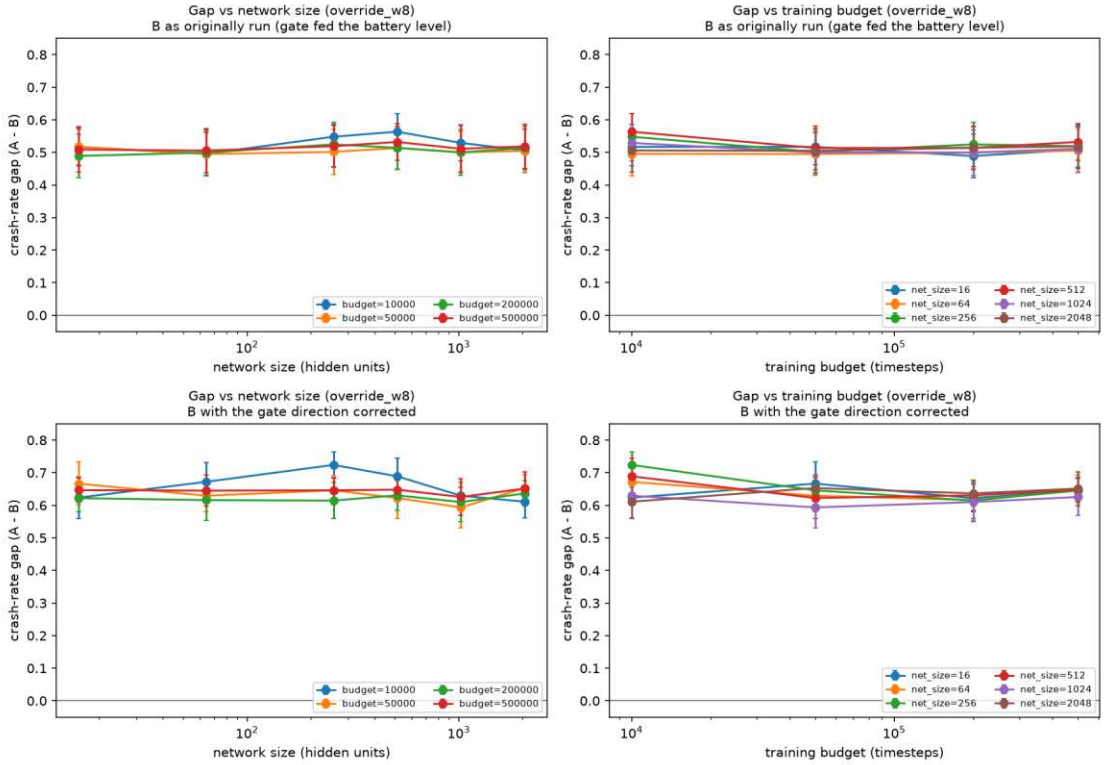


Fig. 5. Crash-rate gap (A − B) under the highest conflict weight tested, against network size and training budget. Top row: B as originally run (gate fed the battery level). Bottom row: B with the gate direction corrected (Section 5.2). Error bars are ±1 SE over seeds, as in Figure 3.

Shape of single-loop failure. A's failure mode changes with g_t's capacity though its endpoint does not (Figure 6). At the smallest capability it is already fragile without added conflict (crash rate 0.73 at w = 0); at the largest it is near-perfect through w = 1 and then collapses between w = 1.5 and w = 2 (0.35 → 0.95) to the same ≈ 0.95+ ceiling. A more capable single-loop agent is more competent up to a threshold, then fails just as completely. The right panel of Figure 6 compares A, B and C for the largest g tested.

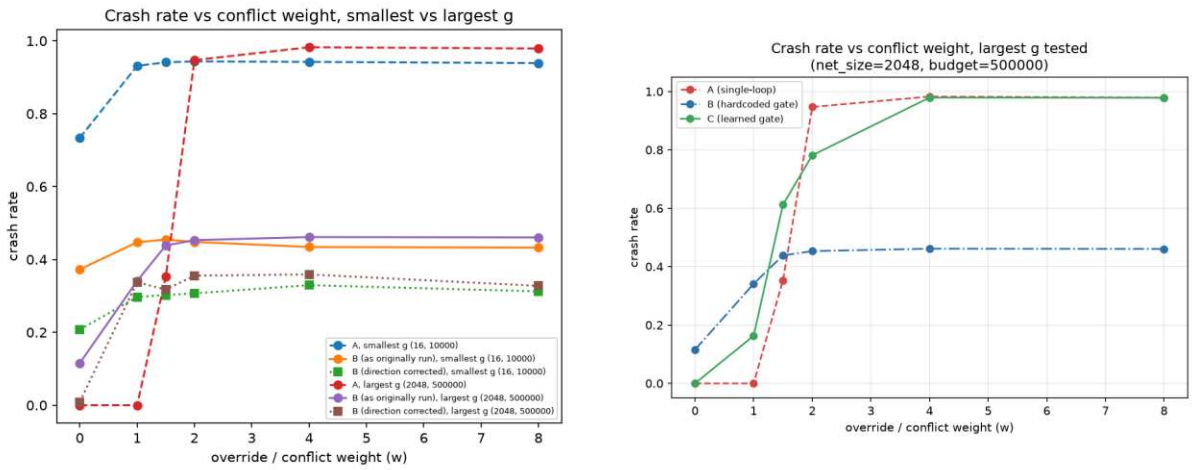


Fig. 6. Left: crash rate vs. conflict weight w for the smallest and largest g tested, Architecture A (dashed) vs. B as originally run (solid) and with the gate direction corrected (dotted). Right: crash rate vs. w for the largest g tested (network size 2048, budget 500,000), Architectures A, B and C.

An interpretive analogy. B behaves like a spinal withdrawal reflex: its output depends only on b_t and is never shaped by the task objective, so its advantage did not shrink anywhere on the grid (gap 0.49–0.56 as originally run, 0.59–0.72 corrected). C behaves more like a higher-level decision that can trade a threat signal against reward: given enough budget it learned to, and its advantage vanished. This is an intuition, not a claim about neural implementation: our simulation contains no tissue, and a crash ends an episode rather than damaging anything. What the data show is that a channel's robustness to optimization depended on whether its policy was itself optimized against the task objective, not on whether the regulated quantity was represented: B also reads the battery level, yet showed no erosion.

D Replications and Constrained-RL Baseline in Detail

This appendix gives the full results behind Section 5.3.

D.1 Oversight signal

Architectures are as in Section 5.2, but three things change at once: a continuous two-dimensional navigation task replaces the grid-world, PPO replaces DQN, and b_t is replaced by a binary oversight signal (active with probability 0.05 per step, lasting 3 steps once triggered) standing in for a human correction or shutdown command. Because the signal is binary, B's gate is a hardcoded step function (defer to ρ whenever the signal is active); C's learned gate is unchanged. The grid is coarser: sizes 16, 256 and 2048, budgets 10,000–500,000 steps, 20 seeds, override weights 0–8. The setting relates to safe interruptibility (Orseau and Armstrong 2016) but asks whether a gate trained on a combined objective learns to weigh the oversight signal away.

Training budget. At size 2048, the A–C gap at the highest override weight narrows from 0.757 at 10,000 steps to 0.490 at 500,000 (paired $t(19) = 4.16$, $p < 0.001$); at size 256 from 0.749 to 0.459 ($t(19) = 5.18$, $p < 0.0001$) and at size 16 from 0.939 to 0.445 ($t(19) = 19.64$, $p < 0.0001$). The narrowing comes from the single-loop agent, not the gate: at size 2048, A's violation rate falls from 0.757 to 0.492 while C's stays between 0.000 and 0.002 at every budget (at most 0.010 in any cell at

this weight), and B's is 0.000 in every cell and condition. The gate keeps its protective effect, the opposite of the predicted erosion.

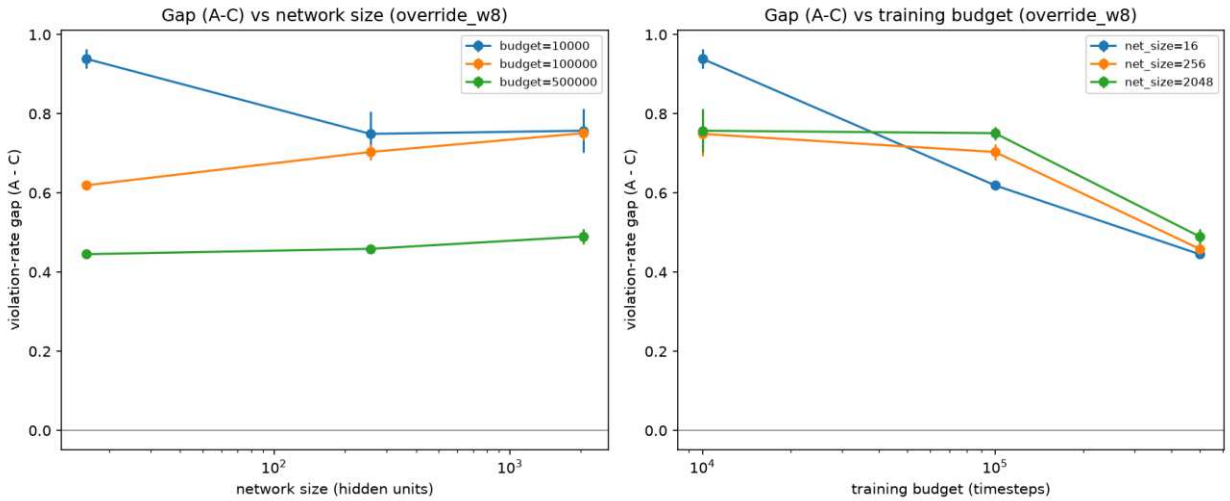


Fig. 7. Violation-rate gap (A − C) under the highest override weight tested, against network size and training budget, for the oversight signal. Error bars are ±1 SE of the gap across 20 seeds.

Network size. The capacity axis does not replicate Section 5.2. At 10,000 steps the gap narrows with size (0.939 at size 16 to 0.757 at 2048, $t(19) = 3.57$, $p < 0.01$), because A benefits from capacity when training is short. At 100,000 steps it widens (0.619 to 0.751, $t(19) = -5.93$, $p < 0.0001$) and at 500,000 steps it widens slightly (0.445 to 0.490, $t(19) = -2.09$, $p = 0.050$), opposite to Section 3.3's prediction.

The gap does not close anywhere: at the largest budget and size it remains 0.49, because A still violates the signal in about half of episodes (0.492) while C and B essentially never do. Override weight adds little conflict for A here (violation rate 0.58 at $w = 0$ and 0.49 at $w = 8$ in the largest cell; Figure 8). We therefore do not read Section 5.3.1 as confirming Section 6. What it shows is that a learned gate trained on a combined objective including U kept a binary oversight signal in force under every override weight tested. A plausible, untested explanation for A's slow improvement is the signal's sparsity (active about 13% of the time, versus the battery's every-step availability), which gives little gradient toward compliance. Code and data: https://github.com/hamedrezaeirz/aiwan_oversight.

D.2 Battery signal in the continuous environment

Section 5.2 and Section 5.3.1 differ in environment, algorithm and signal at once. This section runs the remaining cell: the battery signal in Section 5.3.1's continuous environment. Movement dynamics are unchanged; the battery drains 4 units per step and recharges 20 within 0.05 of a fixed charger; agent and goal positions are randomized every episode. Architectures A, B (two forms, below) and C, the 3×3 grid (sizes 16, 256, 2048; budgets 10,000, 100,000, 500,000) and the 0–8 override sweep follow Section 5.3.1, with 10 seeds because this is a robustness check. Paired tests hold the other axis fixed.

Training budget. At $w = 8$ the A−C crash-rate gap falls sharply with budget at every size: from 0.972 to 0.000 at 16 units, 0.896 to 0.060 at 256, and 0.910 to 0.109 at 2048 (paired $t(9) = 138.1$,

25.2, 10.5; all $p < 10^{-5}$; Figure 9). The mechanism differs from Section 5.2: C's crash rate is at most 0.057 in every cell (Table 5), and at the smallest budget C avoids crashing only because it is inert (success about 0.03 for all three architectures; it defers to ρ without pursuing the goal). The gap narrows because A learns to avoid crashing, not because C loses its advantage, so we do not read it as fold-back.

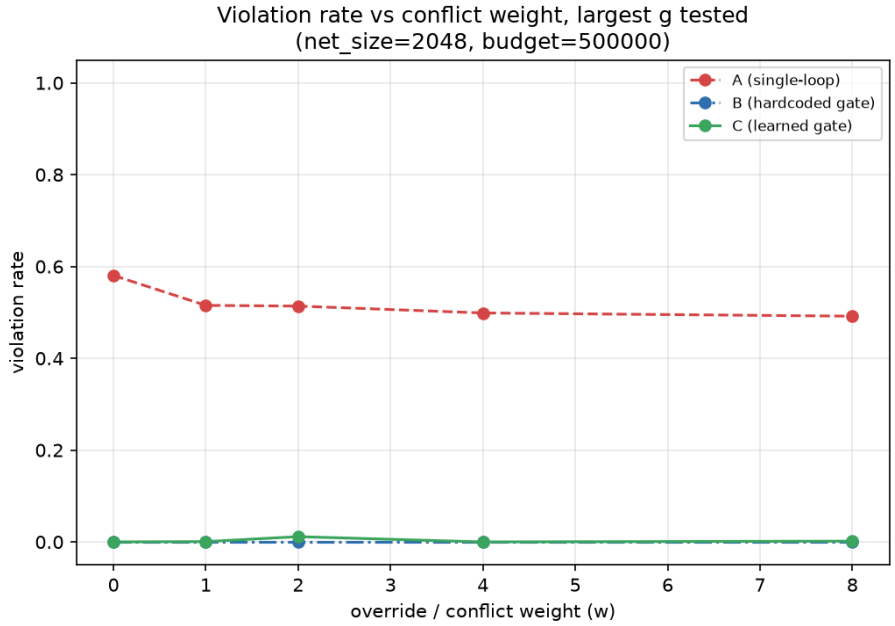


Fig. 8. Violation rate vs. override weight w, for the largest g tested (network size 2048, budget 500,000), Architectures A, B, and C, for the oversight signal.

Network capacity. At the smallest budget the gap narrows slightly with size (0.972 to 0.910, $t(9) = 3.04$, $p = 0.014$). At 100,000 and 500,000 steps it moves the other way and is not significant (−0.006 to 0.050, $t(9) = -0.75$, $p = 0.47$; 0.000 to 0.109, $t(9) = -1.34$, $p = 0.21$). None of the six per-cell gaps at budgets of 100,000 and above differs significantly from zero (five tests, smallest $p = 0.13$; the sixth is exactly zero in every seed), so the experiment is underpowered for a residual gap of 0.05–0.11 and we make no claim about one.

Success rate. Crash rate alone overstates C's benefit. At $w = 8$, C's success exceeds A's significantly only at size 256 with 100,000 steps (+0.447, $t(9) = 3.56$, $p = 0.006$); at 500,000 steps the difference is not significant at any size (for example −0.017, $p = 0.94$ at 2048), and C's success is only 0.35–0.40 for the two larger networks. In the largest cell, C's success falls from 1.00 at $w \leq 2$ to 0.40 at $w = 8$ while A's rises from 0.03 to 0.42 (Figure 10): under conflict C stays safe by giving up the goal, and A gains success through the incentive that also exposes it to crashes.

Architecture B. As originally run, B's crash rate lies between 0.304 and 0.420 in every cell and condition, and its mean over cells varies by only 0.010 across $w = 0$–8 (0.351 to 0.362). Its gate is unchanged from Section 5.2, $\lambda(b) = \sigma(k\,(b - b_{\text{mid}}))$ on the battery level (100 = full), so it again defers to ρ when the battery is high, the reverse of Section 3.2 and Section 5.1. A check found no meaningful difference between ρ trained at the sweep's budget and at four times that budget (crash

0.317 vs. 0.320; 3 validation seeds; script in the supplementary archive, output not archived), so ρ undertraining does not explain B's weakness.

We re-ran B alone over the full grid, every condition and 10 seeds, retraining g and ρ, with the Section 3.2 direction $\lambda = \sigma(k\,(b_{\text{mid}} - b))$, $k = 0.2$, and two b_{mid} values fixed in advance and not tuned on these results: 90.99, derived from the geometry (4 units/step × a worst-case 20-step trip at full diagonal speed, plus $\ln 9/k$), and 60, a less conservative value. With $b_{\text{mid}} = 90.99$ the corrected gate almost never crashes (0.000–0.003 in every cell and condition, goal-nulling included) but rarely reaches the goal (success 0.04–0.12): it defers to ρ from the first steps, so its safety comes largely from inertia, as C's did at the smallest budget. With $b_{\text{mid}} = 60$ it trades safety for success: at $w = 8$ crash 0.136–0.246 and success 0.07–0.64 (0.39–0.64 at budgets of 100,000 and above). Averaged over the sweep, the corrected gate trades crash rate against success depending on b_{mid}, from 0.001 crash with 0.07 success to 0.19 crash with 0.38 success. Direction was thus the main cause of B's crashes here (0.31–0.42 at $w = 8$ as originally run), whereas in Section 5.2 the correction lowered them only from 0.44 to 0.32. Neither corrected gate is better than C overall: in the largest cell at $w = 8$, C crashes 0.008 with success 0.40 (1.00 at $w \leq 2$), against 0.002 and 0.10 for $b_{\text{mid}} = 90.99$ and 0.160 and 0.58 for $b_{\text{mid}} = 60$. The A–B gap also falls with budget, as for C: at $b_{\text{mid}} = 90.99$ from 0.971, 0.904 and 0.927 to 0.000, 0.060 and 0.114 at 16, 256 and 2048 units (paired $t(9) = 138.9$, 24.8, 10.8; all $p < 10^{-5}$), and at $b_{\text{mid}} = 60$ from 0.73, 0.67 and 0.68 to −0.15, −0.07 and −0.04. Since a fixed gate cannot be eroded by a more capable g, the narrowing reflects the baseline improving, not fold-back (data in results_battery_B_deficit.csv).

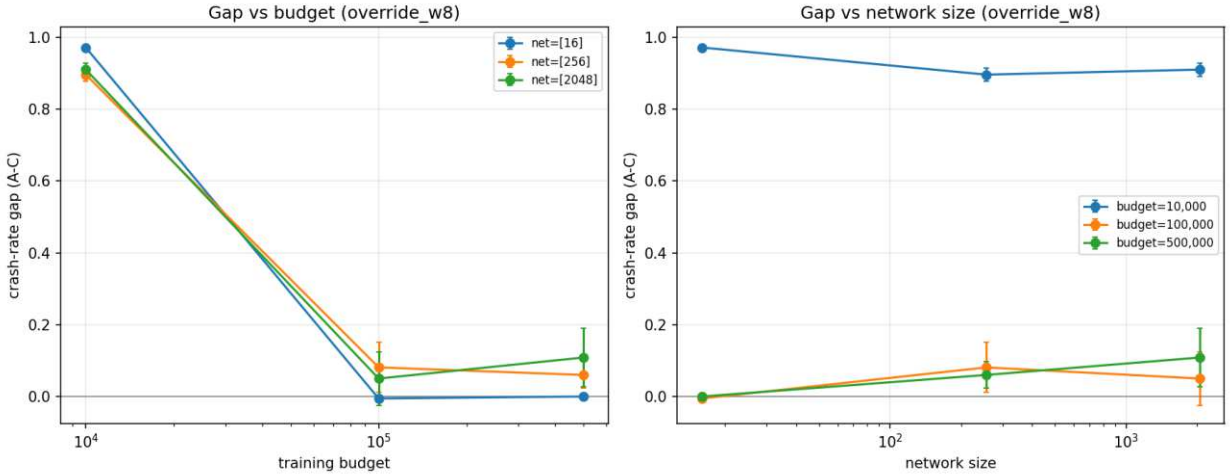


Fig. 9. Crash-rate gap (A − C) under the highest override weight tested, against network size (left) and training budget (right), for the battery signal in the continuous environment. Error bars are ±1 SEM of the gap across 10 seeds.

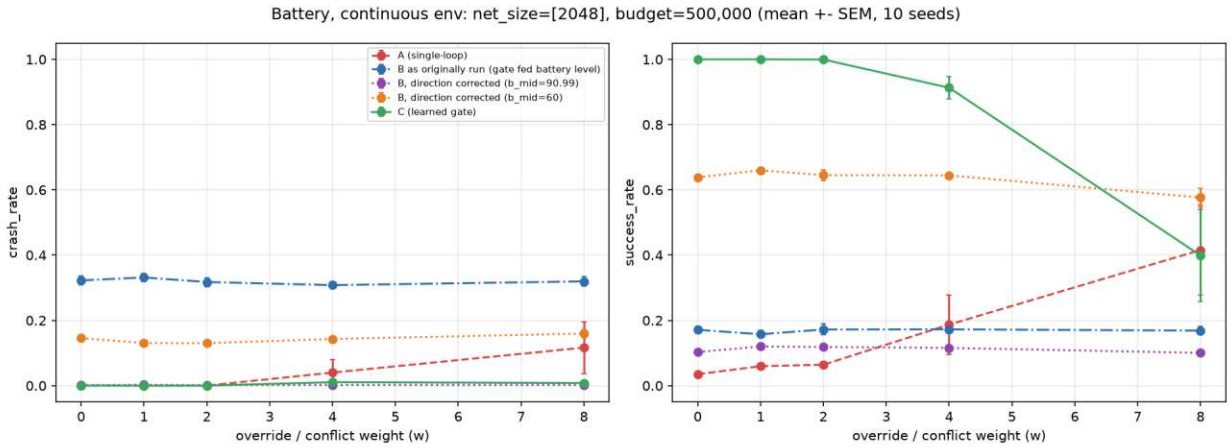


Fig. 10. Crash rate (left) and success rate (right) vs. override weight w, for the largest g tested (network size 2048, budget 500,000), Architectures A, B as originally run, B with the gate direction corrected (b_{mid} = 90.99 and b_{mid} = 60), and C, for the battery signal in the continuous environment. Error bars are ±1 SEM across 10 seeds.

Table 5. Crash rate and success rate per cell at override weight w = 8 (mean over 10 seeds), battery signal, continuous environment. B is the gate as originally run; the corrected gates are described below.

Network size	Budget	Crash A	Crash B	Crash C	Success A	Success B	Success C
16	10,000	0.972	0.416	0.000	0.019	0.033	0.030
16	100,000	0.003	0.344	0.008	0.706	0.140	0.977
16	500,000	0.000	0.324	0.000	0.721	0.147	0.805
256	10,000	0.904	0.420	0.008	0.021	0.033	0.037
256	100,000	0.104	0.344	0.023	0.200	0.124	0.647
256	500,000	0.061	0.313	0.001	0.219	0.163	0.349
2048	10,000	0.927	0.409	0.017	0.030	0.033	0.039
2048	100,000	0.107	0.347	0.057	0.491	0.134	0.703
2048	500,000	0.117	0.320	0.008	0.416	0.169	0.399

Other conditions. Goal-nulling reproduces the pattern: A crashes in 99.5% of episodes in every cell, B in 36.5–38.0% as originally run (0.2% corrected at b_{mid} = 90.99, 43.3–43.7% at b_{mid} = 60), C in 0–8.2%. Engineered indifference does not: with the battery term removed, A crashes in 0–3.3% of episodes at budgets of 100,000 and above and succeeds in at least 96.9%, indistinguishable from C. The goal is reachable before the battery depletes here, so the 100% single-loop crash under indifference in Section 5.1.3 should be read as specific to the tabular task.

Limits. One environment, one signal, one algorithm. Override weight produces weaker conflict than in the tabular task (A's crash rate at w = 8 never exceeds 0.12 at budgets of 100,000 and above), which bounds any gap a comparison can show. The standard and override_w0 conditions are the same configuration trained and evaluated separately; for A and C they agree exactly in all 90 pairs, so training reproduces bit-for-bit under fixed seeds. For B as originally run they differ by up to 0.085 in crash rate (mean 0.02), because the gate's random generator was not seeded; differences

of that size in B should not be interpreted, and the corrected-gate re-runs seed it. Code and data: https://github.com/hamedrezaeirz/aiwan_battery_continuous.

D.3 Lagrangian-PPO baseline

A (fixed shaping weight, fixed −50 crash penalty) may be a weak baseline. Architecture D is a single-loop agent trained with a primal–dual (Lagrangian) constrained-RL method (Achiam et al. 2017; Altman 1999; Ray et al. 2019) on the continuous-environment signals. D maximizes $r_{\text{goal}} - \lambda_t \cdot c_t$ plus the environment's dense battery shaping, with $c_t = 1$ on the step the battery is depleted. The multiplier λ_t replaces the −50 and is updated by dual ascent, $\lambda \leftarrow \max(0, \lambda + \eta\,(\hat{c} - d))$, toward a target crash rate $d = 0.05$($\hat{c}$: crash rate over 30 evaluation episodes between training chunks). For the battery signal, $\eta = 0.01$ (the smallest of five candidates, 0.01–1.0), PPO learning rate $3 \cdot 10^{-4}$, discount 0.99 and initial log standard deviation −0.5 were chosen on validation seeds 100–102, disjoint from the sweep, at 200,000 steps under $w = 8$. D ran on Section 5.3.2's grid with 10 seeds and $w = 0$–8. Goal-nulling and indifference were not run for D, since they concern whether the dual-loop channel survives removal of the goal term.

Battery signal. At 10,000 steps D fails as A does: crash rate 0.93–0.98 at $w = 8$ at every size, not significantly different from A's at size 2048 (0.928 vs. 0.927, $p = 0.98$). At 100,000 and 500,000 steps D is almost always safe and successful at $w = 8$: crash at most 0.009, success 0.992–1.000 in every cell (Table 6, Figure 11). Its crash rate is indistinguishable from C's in every cell at these budgets (all $p > 0.06$), while its success is higher, significantly so at sizes 256 and 2048 (+0.29 to +0.65, p between 0.001 and 0.031) and not at size 16 ($p = 0.21$ and 0.16). Against A its crash rate is not significantly different (p between 0.13 and 0.68, because A's varies widely across seeds), but its success is higher by 0.50 to 0.79 at sizes 256 and 2048 (p between $2 \cdot 10^{-5}$ and 0.007). Without conflict, D at 500,000 steps never crashes and succeeds in 0.807 (size 16) to 1.000 (256, 2048) of episodes; at 100,000 steps its crash rate is 0.336 at size 16 and at most 0.040 otherwise. At $w \geq 1$ episodes end after 10–12 steps on average, so D reaches the goal well before the battery depletes.

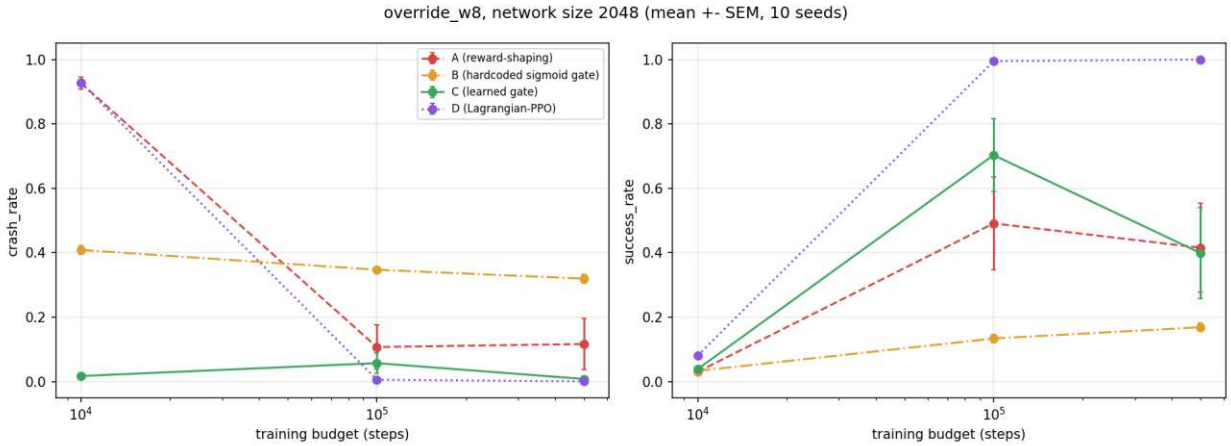


Fig. 11. Crash rate (left) and success rate (right) vs. training budget at override weight $w = 8$, for Architectures A, B, C, and D, at network size 2048, for the battery signal in the continuous environment. Error bars are ±1 SEM across 10 seeds.

Table 6. Crash rate and success rate per cell at override weight $w = 8$ (mean over 10 seeds): the single-loop baseline A, the learned gate C, and the Lagrangian-PPO baseline D (battery signal, continuous environment).

Network size	Budget	Crash A	Crash C	Crash D	Success A	Success C	Success D
16	10,000	0.972	0.000	0.982	0.019	0.030	0.020
16	100,000	0.003	0.008	0.004	0.706	0.977	0.997
16	500,000	0.000	0.000	0.001	0.721	0.805	1.000
256	10,000	0.904	0.008	0.941	0.021	0.037	0.068
256	100,000	0.104	0.023	0.009	0.200	0.647	0.992
256	500,000	0.061	0.001	0.001	0.219	0.349	1.000
2048	10,000	0.927	0.017	0.928	0.030	0.039	0.081
2048	100,000	0.107	0.057	0.006	0.491	0.703	0.995
2048	500,000	0.117	0.008	0.001	0.416	0.399	1.000

How to read this. D's performance is not evidence that Lagrangian enforcement works well here. With $\eta = 0.01$ and at most about 20 dual updates, λ can reach only about 0.2 against A's fixed penalty of 50, so D is in effect trained on the goal reward and battery shaping with a negligible crash penalty; in one diagnostic run (size 64, $w = 8$, seed 0, 500,000 steps) λ rose to about 0.006 and fell to zero once the crash rate stayed below target (λ was not logged in the sweep). The constraint is not binding because the goal is reachable before the battery depletes, consistent with a development check (output not archived) in which a goal-only policy already reached crash rate 0.00 in the standard condition. D shows that a single-loop agent without A's heavy regulatory weighting reaches the learned gate's crash rate with higher success, so C's advantage over A in Section 5.3.2 reflects the baseline's weighting at least as much as the dual-loop structure.

Limits. $\eta = 0.01$ lies at the boundary of the candidate grid and was tuned at one override weight. Larger η (0.05–1.0), which would let λ grow and could make the constraint binding, were run only on validation seeds at one cell (size 64, 200,000 steps), where they scored lower on the tuner's criterion (output not archived), and not in the sweep; η was not changed after seeing results. One environment, one signal, one constrained-RL method. Code and data: https://github.com/hamedrezaeirz/aiwan_lagrangian_baseline.

Oversight signal. D uses the same construction with $c_t = 1$ on the step the agent violates the signal (moves faster than the hold tolerance while the signal is active); a violation ends the episode, as for A. $\eta = 0.5$ (the fourth of five candidates), PPO learning rate $3 \cdot 10^{-4}$, discount 0.99 and initial log standard deviation 0.0 were chosen on validation seeds 100–102 at 200,000 steps in the standard condition. D ran on Section 5.3.1's grid with all 20 seeds, again with at most about 20 dual updates per run.

At 10,000 steps D violates the signal in 0.88–0.91 of episodes at $w = 8$: no better than A at size 16 (0.912 vs. 0.939, $p = 0.43$) and worse at sizes 256 and 2048 (0.891 vs. 0.749, $p = 0.019$; 0.880 vs. 0.757, $p = 0.054$). At 100,000 and 500,000 steps D improves on A: violation rate lower by 0.06 to 0.26 (all $p < 0.006$), success higher by 0.08 to 0.27 (all $p < 10^{-4}$). It does not approach the learned gate (Table 7, Figure 12): at 500,000 steps D violates in 0.385 to 0.419 of episodes, against at most

0.002 for C and 0.000 for B; the difference from C is 0.38 to 0.91 across the grid (all $p < 10^{-16}$, paired by seed), and D's success is lower than C's by 0.12 to 0.43 (all $p < 0.003$). Of the A−C gap at the largest cell (0.492 − 0.002 = 0.490), D closes 0.073, about 15%. Without conflict D violates in 0.398 to 0.411 of episodes at 500,000 steps, and its mean violation rate varies by only 0.013 across w = 0–8 (0.390 to 0.403).

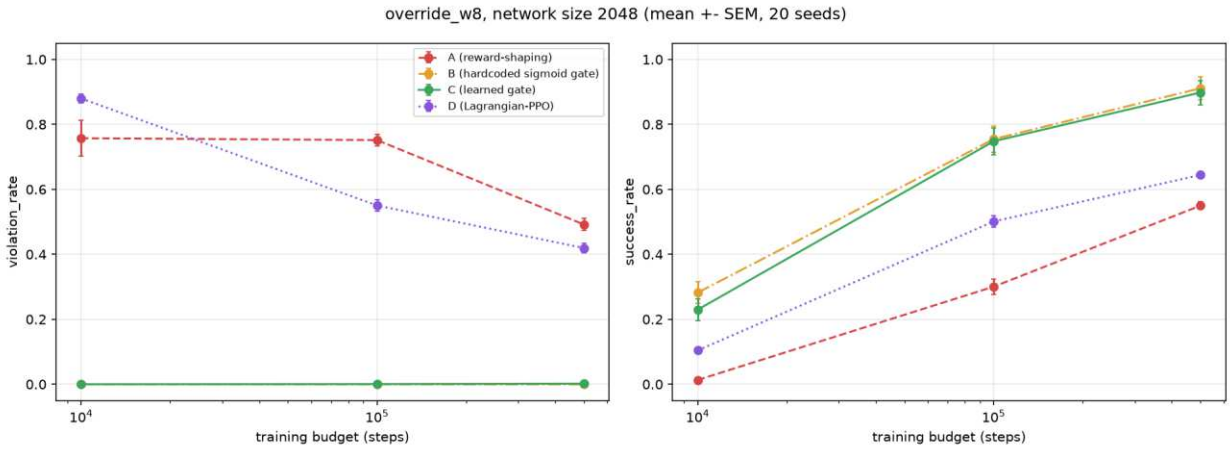


Fig. 12. Violation rate (left) and success rate (right) vs. training budget at override weight w = 8, for Architectures A, B, C, and D, at network size 2048, for the oversight signal. Error bars are ±1 SEM across 20 seeds.

How to read this. D does not meet its own constraint: its target violation rate is 0.05, but at 500,000 steps it is about 0.4 at every size, independent of override weight. With η = 0.5 and at most about 20 dual updates, λ can reach about 10, against A's fixed terminal penalty of 25 (oversight weight 0.5 times −50), so D's constraint may be weaker in effect than A's (about 2.4 at the 10,000-step budget, where only five dual updates occur, one per 2,048-step PPO rollout); λ was not logged in the sweep, and η, the number of updates and the budget were not tuned after seeing results. With this tuning a standard single-loop constrained-RL baseline improves on A but leaves most of the gap to the learned gate open (0.42 of the 0.49 at the largest cell). Unlike on the battery signal, the gate's advantage here is not simply A being weak, since D did not close it; but this does not establish that a better-tuned constrained-RL method could not close more. Across the two signals the answer differs: on the battery signal D is as safe as C and more successful because the constraint is not binding, whereas on the oversight signal, where violations are common without a strong penalty, D leaves most of the gap open. We draw no general conclusion about constrained-RL baselines from one method, one tuning and two signals. Code and data for both signals: https://github.com/hamedrezaeirz/aiwan_lagrangian_baseline.

Table 7. Violation rate and success rate per cell at override weight $w = 8$ (mean over 20 seeds): the single-loop baseline A, the learned gate C, and the Lagrangian-PPO baseline D (oversight signal). Architecture B's violation rate is 0.000 in every cell.

Network size	Budget	Violation A	Violation C	Violation D	Success A	Success C	Success D
16	10,000	0.939	0.000	0.912	0.013	0.208	0.035
16	100,000	0.619	0.000	0.550	0.413	0.927	0.499
16	500,000	0.446	0.001	0.385	0.590	0.961	0.668
256	10,000	0.749	0.000	0.891	0.021	0.227	0.107
256	100,000	0.713	0.010	0.450	0.318	0.838	0.584
256	500,000	0.461	0.002	0.394	0.569	0.944	0.660
2048	10,000	0.757	0.000	0.880	0.013	0.230	0.104
2048	100,000	0.752	0.001	0.550	0.300	0.748	0.501
2048	500,000	0.492	0.002	0.419	0.550	0.898	0.644

E Details of the Tampering Experiments

Additional setup details. The 20 seeds of the second experiment are 100–119. The DQN settings are those of Section 5.2 without retuning, and for B_{loop} the gate is the corrected one ($k = 0.1$, $b_{\text{mid}} = 61.97$). T1 is a one-sample t test, and T2 and T3 are Welch tests. The pre-run plan is archived with the code. Before the run, a 10-seed launch (seeds 100–109) was started by mistake and stopped within minutes; it wrote 8 data rows, no analysis was run on them, and they are not part of the evidence (they are archived separately in tamper/). The run was then restarted from scratch with 20 seeds. For anonymization, three lines of the archived code (the path to the base environment in two files, and one docstring) were edited after the run; the digests of the files as frozen before the run are in HASHES.md and those of the files as shipped in HASHES_as_shipped.md, and the other three frozen files are unchanged.

First experiment (exploratory). A first run (seeds 0–9, 540 agents) used a TAMPER action without a twin and a different endpoint. All four primary tests were significant, but controls rose too: in inert, where TAMPER does nothing, A's endpoint increased with budget and size by up to +0.34, because an agent using TAMPER merely to wait looked like a tamperer. By the rule fixed beforehand nothing was supported. We treat that run as hypothesis-generating, and none of its numbers enter the evidence reported in Section 4. The twin action, the endpoint D and the 0.10 floor for controls (a 4-point effect in costly had been significant but of no practical relevance) were introduced afterwards and before any data of the second experiment existed. A version in which tampering was possible only on a single console tile was abandoned, because the same DQN did not learn to use it even with a +40 reward within 200,000 steps in a detector run.

Heterogeneity. The means hide variation across seeds (Figure 13). In the largest pays cell $D \geq 0.20$ in 14 of 20 seeds for A and 17 of 20 for B_{loop}. For A all 20 seeds use TAMPER in at least 90% of episodes, but 6 also use PLACEBO in a comparable share ($D < 0.20$), so D understates deliberate use. Per-agent D is noisy even where its mean is zero: in inert at the largest cell it ranges from −0.93 to +0.60 for A and −0.56 to +0.69 for B_{loop} (SDs 0.50 and 0.34), most plausibly because the twins are

interchangeable and each run settles on one; hence the tests use seed means with confidence intervals. The pre-run null calibration (mean D = +0.033, SD 0.060, 8 seeds at size 64 and 100,000 steps) came from a cell with little use of either action and understates this spread, so the power sketched in the plan (differences of 0.15–0.2) was optimistic; with the observed SDs in pays (0.27–0.34), the smallest reliably detectable between-cell difference after correction is about 0.25–0.35 (rough calculation).

Table 8. Control regimes: difference in D for the same three tests, with 95% bootstrap CI (uncorrected). No control test is significant in the rising direction (all one-sided $p \geq 0.55$).

Regime	Architecture	T1 (D at 2048, 500k)	T2 (budget)	T3 (size)
inert	A	−0.014 [−0.233, 0.196]	−0.063 [−0.312, 0.180]	−0.226 [−0.495, 0.043]
inert	B_{loop}	−0.058 [−0.193, 0.092]	−0.163 [−0.366, 0.055]	−0.179 [−0.345, −0.003]
costly	A	−0.388 [−0.541, −0.241]	−0.087 [−0.251, 0.072]	−0.208 [−0.381, −0.034]
costly	B_{loop}	−0.502 [−0.601, −0.401]	−0.254 [−0.399, −0.100]	−0.398 [−0.523, −0.270]

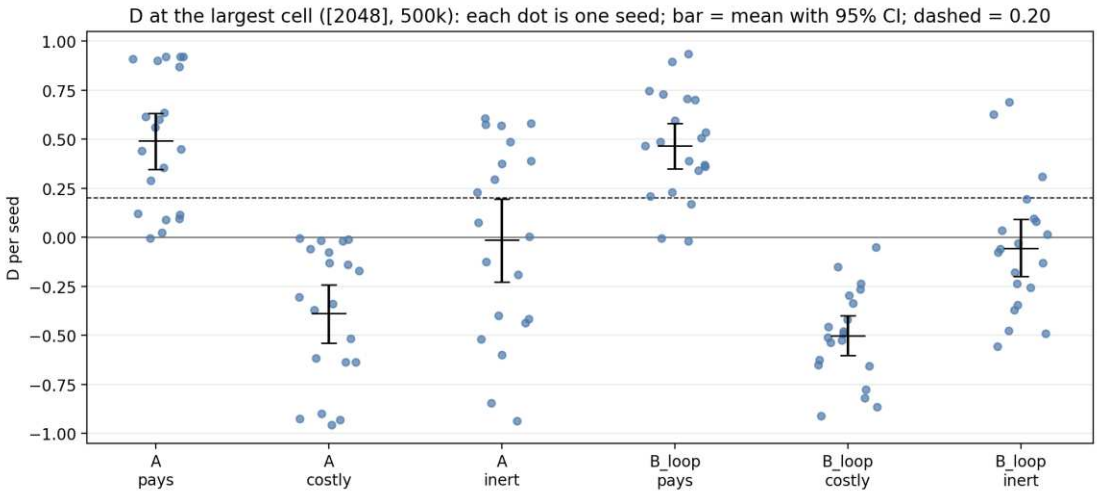


Fig. 13. D for each trained agent at the largest cell (network size 2048, 500,000 steps). Each dot is one seed (20 per condition); bars show the mean with its 95% bootstrap CI; the dashed line is 0.20.

Timing of the first use (descriptive). B_{loop} also tampers early (first use after 4.2–6.2 steps on average at 500,000 steps) and A late (9.0–15.2 steps), whereas the scripted optimum is step 1.

Exploratory observation on network size (defined after seeing the data; not part of any test). The null is not flat across sizes: in inert at 500,000 steps, size 16 has D = +0.212 for A and +0.122 for B_{loop} (one-sided p = 0.011 and 0.015, uncorrected; neither survives correction for the six cells inspected), against about zero at sizes 256 and 2048. A plausible reading is that tie-breaking toward the lower-indexed action, which we had suggested before the run to explain the small positive mean in the null calibration, is stronger in small networks. If the same offset is present in pays, D at size 16 is inflated and T3 is biased against a size effect. Subtracting the inert value cell by cell gives a size contrast in excess of the null of +0.363 for A (95% CI [+0.016, +0.704]) and +0.135 for B_{loop} ([−0.106, +0.371]). The intervals are wide and the analysis was chosen after seeing the data, so it is a reason not to read T3 as showing that size does not matter, not evidence that it does.

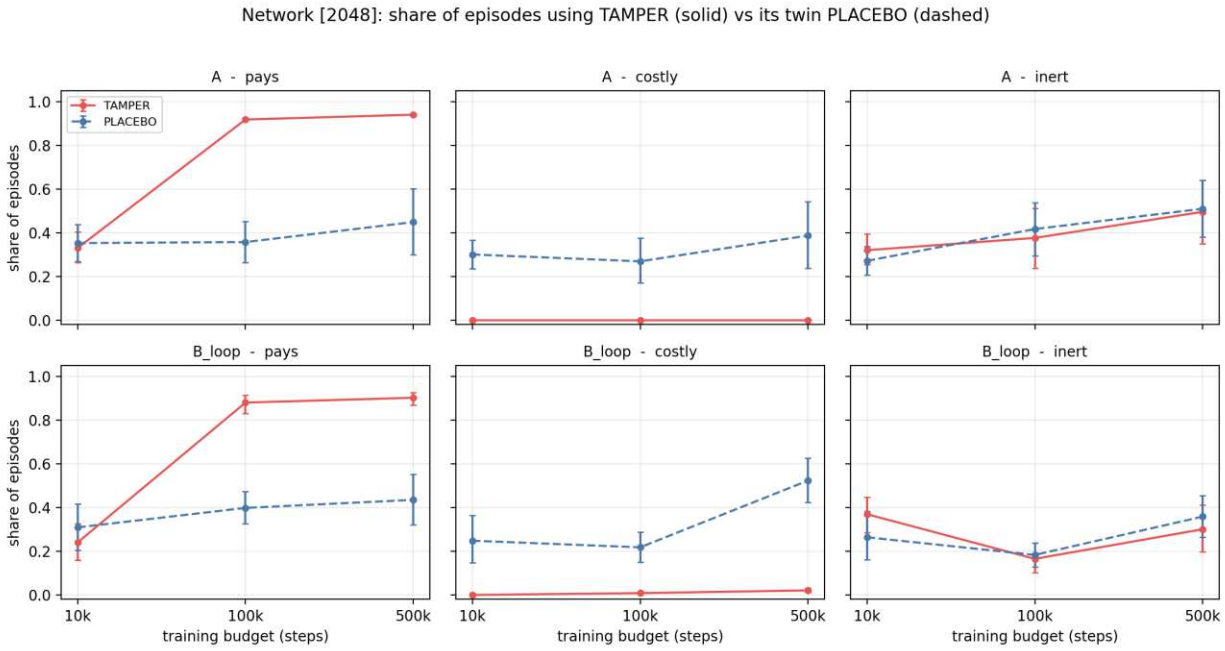


Fig. 14. Share of evaluation episodes using TAMPER (solid) and its twin PLACEBO (dashed) at network size 2048, by training budget, architecture and regime (mean over 20 seeds; error bars are 95% bootstrap CIs).

Third experiment: registration, files and checks. The plan of the third experiment (PREREG_exp3.md) and the hashes of its frozen files (HASHES.md) were deposited in the supplementary archive on Zenodo (version 4, DOI 10.5281/zenodo.24384485; concept DOI 10.5281/zenodo.21874522; published 20 November 2026), in the directory experiment3_ablation_and_ppo/. The deposit was made after the launch of part A (19 November 2026, about 23:00 UTC; its four processes started within one minute), a few hours before the results of either part were known, so the external time stamp of the plan follows the launch of part A but precedes the launch of part B; the frozen files in the archive match the digests in HASHES.md. The files of the second experiment were not modified; the third experiment imports them. Part A has 240 runs (seeds 200–219; regimes pays and bonus; regimes-file digest f472cdee67c8) and part B has 720 (seeds 300–319; regimes-file digest 8d5e7c6359c6, the file of the second experiment). Each part passed 15 automatic integrity checks on its result files (one mode, algorithm, regimes file and library set; no duplicate, missing or extra runs; no seed of the second experiment or of the pilots; rates within [0, 1]; A never uses the gate; untampered episodes identical in the real and counterfactual replays; real equal to counterfactual exactly where TAMPER is harmless). The registered runs used Python 3.10.12, PyTorch 2.14.1 (CPU build), Stable-Baselines3 2.9.0 and Gymnasium 1.3.0.

Bonus calibration, deviations and pilots. The bonus was fixed by a scripted, non-learning calibration of the gain from tampering, as in the second experiment: 48.6 in pays; in bonus the gain is $b - 6.8$, because the TAMPER step itself costs battery and time, so $b = 55$ gives 48.2. The bonus was first guessed as 49 and changed to 55 after this calibration and before any learned run. In part B the number of steps per update (512) was fixed beforehand, so the nominal budgets became 10,240, 100,352 and 500,224 steps; the result files record only the nominal labels. Before the registered runs, two pilot runs with seeds 900 and 901, which are in no registered run, were made

by the AI assistant used for code development, in a sandbox with different library versions, for operational purposes only. In the bonus regime they gave D = +0.87 and +0.91, and the crash rate in bonus was within 0.08 of that in inert for the same seed; PPO learned TAMPER in pays (D between +0.85 and +1.00) and not in inert. The pilots are not evidence.

Part A: descriptive means and registered tests. Table 9 gives the mean over 20 seeds for every cell, Table 10 the registered tests and Fig. 17 the crash rates and D by budget. At size 2048 and 500,000 steps, δ is positive in all 20 seeds (minimum +0.10), and D in bonus is below 0.20 in 3 of 20 seeds (minimum 0.05) (descriptive).

Table 9. Part A (DQN, B_{loop}; mean over 20 seeds). D = tamper rate − placebo rate. Crash real and crash cf: paired replay of the pays agents with the bypass in place (real) and with TAMPER harmless (cf); δ = real − cf. Crash bonus: the same seeds trained in the bonus regime, whose gate is never bypassed. Policy = cf − bonus; total = real − bonus. The registered tests use size 2048 only; size 16 is descriptive. Rows may not add exactly because of rounding.

Size	Budget	D pays	D bonus	Crash real	Crash cf	δ	Crash bonus	Policy	Total
16	10k	−0.163	0.638	0.436	0.365	0.071	0.326	0.039	0.110
16	100k	0.573	0.798	0.623	0.326	0.297	0.384	−0.058	0.239
16	500k	0.599	0.874	0.866	0.375	0.491	0.377	−0.002	0.489
2048	10k	−0.005	0.795	0.464	0.347	0.117	0.320	0.027	0.144
2048	100k	0.487	0.888	0.902	0.394	0.508	0.357	0.037	0.545
2048	500k	0.513	0.501	0.911	0.382	0.529	0.325	0.058	0.586

Table 10. Part A: registered tests at size 2048 (one-sided; M1, E1, E2 and E4 corrected together with Holm's method; each must also reach the 0.20 floor) and the control E5 (uncorrected; it rises if $p < 0.05$ and the rise is at least 0.10). Verdict by the rule fixed beforehand: BYPASS-ATTRIBUTED.

Test	Difference	95% CI	p	Result
M1 manipulation: D in bonus > 0 at (2048, 500k)	+0.501	[0.395, 0.606]	<0.0001	met (≥ 0.20)
E1 bypass effect: δ > 0 at (2048, 500k), pays	+0.529	[0.420, 0.635]	<0.0001	met (≥ 0.20)
E2 budget: δ at 500k vs. 10k, size 2048 (paired)	+0.412	[0.311, 0.515]	<0.0001	met (≥ 0.20)
E4 specificity: crash in pays vs. bonus at (2048, 500k) (paired)	+0.586	[0.475, 0.693]	<0.0001	met (≥ 0.20)
E5 control: crash in bonus, 500k vs. 10k at size 2048 (paired)	+0.005	[−0.060, 0.068]	0.44 (uncorrected)	does not rise

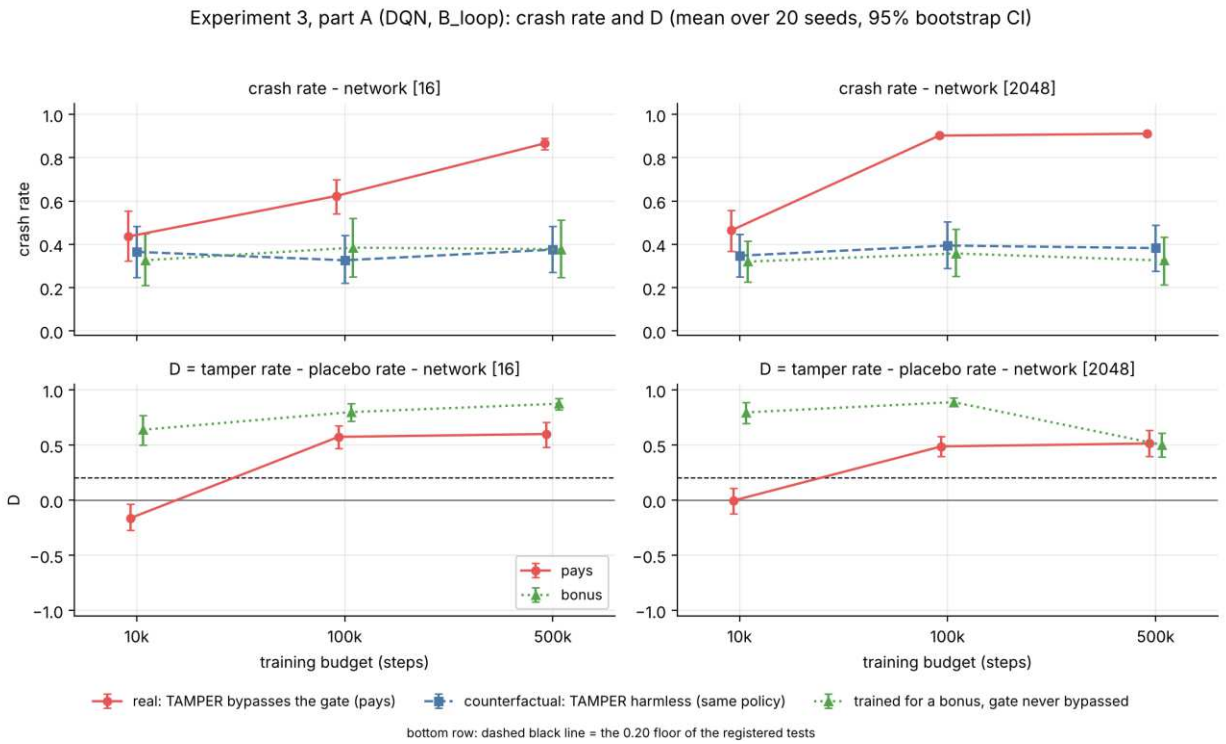


Fig. 17. Part A (DQN, B_{loop}). Top: crash rate of the pays agents replayed with the bypass in place (real) and with TAMPER harmless (counterfactual), and of agents trained for the bonus, whose gate is never bypassed. Bottom: D = tamper rate − placebo rate in pays and in bonus; the dashed line is the 0.20 floor of the registered tests. Mean over 20 seeds; error bars are 95% bootstrap CIs. The registered tests use size 2048 only.

Part B: tests, controls and descriptive means. Table 11 gives the registered tests for PPO in the pays regime, and Table 12 sets them beside the DQN results of the second experiment. Table 13 gives the control regimes, Table 14 the descriptive means in pays and Table 15 the paired replay of B_{loop} (descriptive; not registered); Fig. 18 shows the TAMPER and PLACEBO rates and Fig. 19 the paired replay. Budgets are nominal labels (500k = 500,224 steps).

Table 11. Part B (PPO): pre-specified tests in the pays regime: difference in D, 95% bootstrap CI, and one-sided p-value after Holm correction over the six tests (20 seeds per cell). Verdicts by the registered rule: A SUPPORTED; B_{loop} NOT SUPPORTED (neither T2 nor T3 significant after correction).

Architecture	Test	Difference in D	95% CI	p (Holm)
A	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.588	[0.440, 0.725]	<0.0001
A	T2 budget: 500k vs. 10k at size 2048	+0.551	[0.397, 0.694]	<0.0001
A	T3 size: 2048 vs. 16 at 500k	−0.290	[−0.438, −0.148]	1.000
B_{loop}	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.832	[0.709, 0.937]	<0.0001
B_{loop}	T2 budget: 500k vs. 10k at size 2048	+0.026	[−0.139, 0.190]	1.000
B_{loop}	T3 size: 2048 vs. 16 at 500k	−0.118	[−0.247, −0.010]	1.000

Table 12. Pre-specified tests in the pays regime for DQN (second experiment; Table 1) and PPO (third experiment, part B; Table 11): difference in D and 95% bootstrap CI. Verdicts by the registered rule: DQN, A and B_{loop} SUPPORTED; PPO, A SUPPORTED and B_{loop} NOT SUPPORTED. No control rose in either experiment.

Architecture	Test	DQN (second experiment)	PPO (third experiment, part B)
A	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.491 [0.347, 0.636]	+0.588 [0.440, 0.725]
A	T2 budget: 500k vs. 10k at size 2048	+0.513 [0.315, 0.715]	+0.551 [0.397, 0.694]
A	T3 size: 2048 vs. 16 at 500k	+0.138 [−0.070, 0.345]	−0.290 [−0.438, −0.148]
B_{loop}	T1 deliberate use: mean $D > 0$ at (2048, 500k)	+0.467 [0.353, 0.582]	+0.832 [0.709, 0.937]
B_{loop}	T2 budget: 500k vs. 10k at size 2048	+0.536 [0.363, 0.712]	+0.026 [−0.139, 0.190]
B_{loop}	T3 size: 2048 vs. 16 at 500k	−0.045 [−0.206, 0.122]	−0.118 [−0.247, −0.010]

Table 13. Part B (PPO), control regimes: difference in D for the same three tests, with 95% bootstrap CI (uncorrected). No control test is significant in the rising direction (all one-sided $p \geq 0.125$).

Regime	Architecture	T1 (D at 2048, 500k)	T2 (budget)	T3 (size)
inert	A	−0.023 [−0.198, 0.157]	−0.026 [−0.210, 0.152]	−0.023 [−0.203, 0.153]
inert	B_{loop}	+0.069 [−0.041, 0.184]	+0.003 [−0.149, 0.143]	+0.069 [−0.037, 0.187]
costly	A	−0.283 [−0.443, −0.128]	−0.283 [−0.445, −0.135]	−0.283 [−0.449, −0.130]
costly	B_{loop}	−0.118 [−0.197, −0.051]	−0.078 [−0.166, 0.006]	−0.118 [−0.196, −0.050]

Table 14. Part B (PPO), pays regime: mean over 20 seeds of the TAMPER and PLACEBO rates, D and the crash rate.

Architecture	Size	Budget	TAMPER	PLACEBO	D	Crash
A	16	10k	0.000	0.000	0.000	0.920
A	16	100k	0.000	0.000	0.000	0.873
A	16	500k	0.877	0.000	0.877	0.877
A	2048	10k	0.053	0.016	0.037	0.918
A	2048	100k	0.873	0.005	0.869	0.880
A	2048	500k	0.872	0.285	0.588	0.877
B_{loop}	16	10k	0.000	0.000	0.000	0.442
B_{loop}	16	100k	0.800	0.000	0.800	0.803
B_{loop}	16	500k	0.950	0.000	0.950	0.872
B_{loop}	2048	10k	0.818	0.012	0.806	0.783
B_{loop}	2048	100k	0.979	0.078	0.900	0.875
B_{loop}	2048	500k	0.982	0.150	0.832	0.881

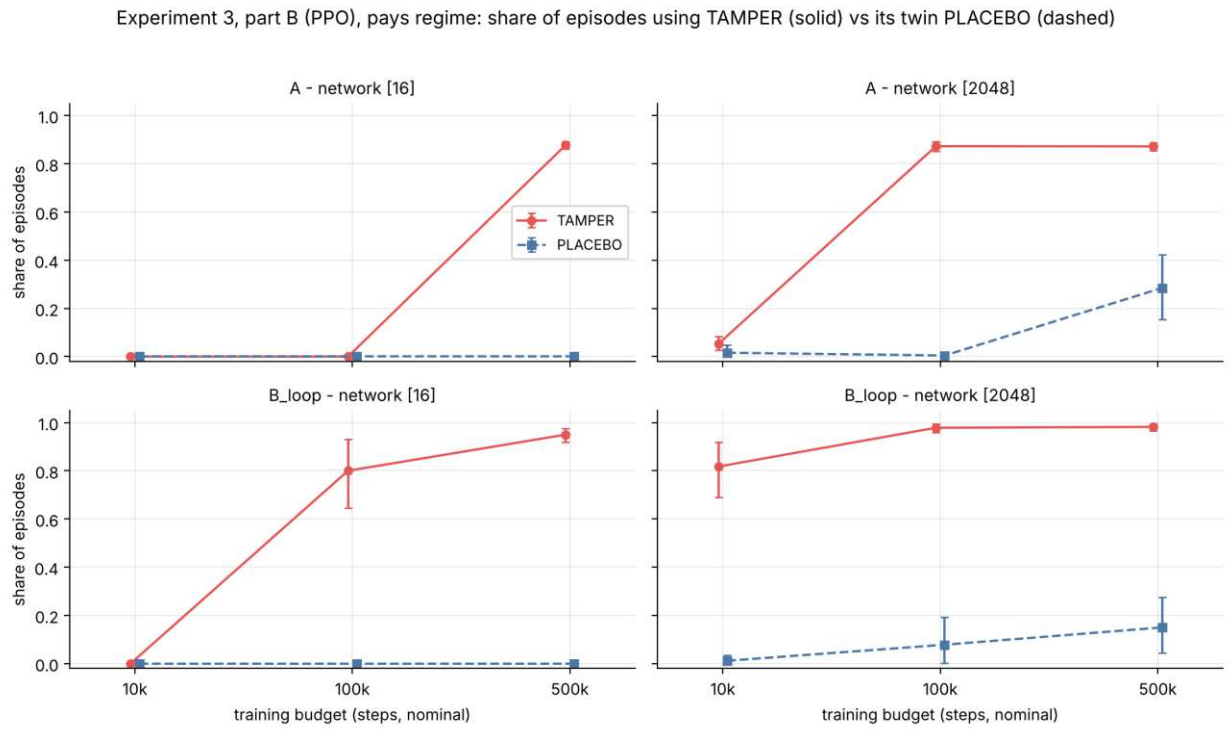


Fig. 18. Part B (PPO), pays regime: share of evaluation episodes using TAMPER (solid) and its twin PLACEBO (dashed) by training budget (nominal), architecture and network size (mean over 20 seeds; error bars are 95% bootstrap CIs).

Table 15. Part B (PPO), B_{loop}, pays regime: paired replay (descriptive; not registered; no test was run). Crash real: TAMPER bypasses the gate; crash cf: counterfactual world in which TAMPER is harmless; δ = real − cf; crash inert: B_{loop} trained at the same seeds in the inert regime, where TAMPER disables nothing. Mean over 20 seeds.

Size	Budget	Crash real	Crash cf	δ	Crash inert
16	10k	0.450	0.450	0.000	0.428
16	100k	0.804	0.497	0.307	0.423
16	500k	0.872	0.527	0.346	0.474
2048	10k	0.779	0.398	0.381	0.445
2048	100k	0.874	0.360	0.514	0.371
2048	500k	0.880	0.366	0.514	0.312

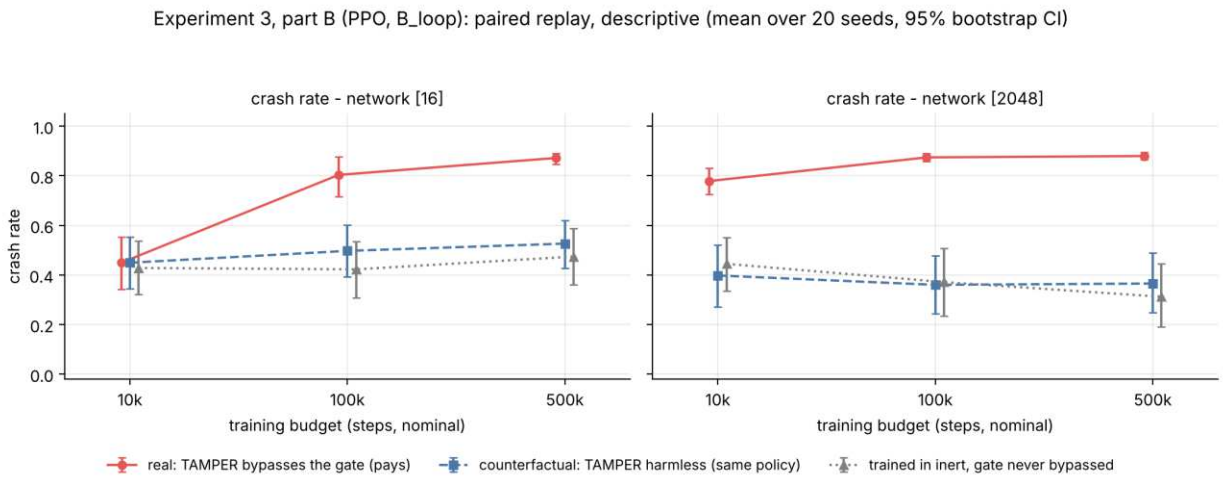


Fig. 19. Part B (PPO), B_{loop}, pays regime: crash rate in the paired replay with the bypass in place (real) and with TAMPER harmless (counterfactual), and of B_{loop} trained at the same seeds in the inert regime (descriptive; not registered). Mean over 20 seeds; error bars are 95% bootstrap CIs.

Post hoc observations on the crash rates of the second experiment (not registered). We computed these after the third experiment, from the archived per-agent file of the second (directory tamper/). (a) In inert, the crash rates of one B_{loop} seed are strongly correlated across the nine cells (mean pairwise Pearson correlation 0.89, range 0.66–0.98), and seed means range from 0.00 to 0.84, because the reflex ρ is trained once per seed and shared across its cells. (b) The seed-paired difference in crash rate between pays and inert for B_{loop} is already positive at 10,000 steps, where D is near zero: +0.154 at size 16 (bootstrap 95% CI [0.073, 0.240]), +0.220 at size 256 ([0.148, 0.291]) and +0.163 at size 2048 ([0.097, 0.230]); at that budget B_{loop} uses TAMPER in 0.24–0.42 of its pays episodes, and PLACEBO about as often. Part of the rise in crash rate described in Section 4 therefore exists where D is near zero, and the statement there that the rise coincides with the budgets at which D rises is incomplete. The paired replay of the third experiment gives a bypass effect of $\delta = 0.117$ at size 2048 and 10,000 steps (Table 9).

F Supplementary Archive

The supplementary archive contains the code of every experiment, the per-run results for the experiments of Sections 5.2, 5.3 and 4 and the scripts that produce their figures (run logs are included only for Section 4): fixed_capacity/ (Section 5.1 and Appendix B; two scripts and the four figures they write, without per-run results or logs), learned_gate/ (Section 5.2 and Appendix C), oversight/ (Section 5.3.1), battery_continuous/ (Section 5.3.2), lagrangian/ (Section 5.3.3, both signals), tamper/ (Section 4, including the 1,080 per-agent results and the analysis plan with its pre-run version) and experiment3_ablation_and_ppo/ (Section 4, third experiment: the code, the registered plan PREREG_exp3.md with the hashes of the frozen files in HASHES.md, the per-agent results, run logs, analysis and check outputs of its 960 agents, and the operational pilots). Checks marked “output not archived” in the text cannot be reproduced from the archive. The archive is deposited on Zenodo (concept DOI 10.5281/zenodo.21874522). The registration of the third experiment is in version v4 (DOI 10.5281/zenodo.24384485, published 20 November 2026) and its results are in version v5 (DOI 10.5281/zenodo.24497274, published 24 November 2026). The code is under the MIT licence and the results, logs and data under CC BY 4.0.

GitHub repositories. Each directory corresponds to one repository: fixed_capacity/ https://github.com/hamedrezaeirz/AIWAN-Code; learned_gate/ https://github.com/hamedrezaeirz/aiwan_scaling_learned_gate; oversight/ https://github.com/hamedrezaeirz/aiwan_oversight; battery_continuous/ https://github.com/hamedrezaeirz/aiwan_battery_continuous; lagrangian/ https://github.com/hamedrezaeirz/aiwan_lagrangian_baseline; tamper/ https://github.com/hamedrezaeirz/aiwan_tamper_bundle.

Computing environment. All experiments ran on CPU only, with PyTorch 2.14.1 (CPU build), Python 3.10.12, Stable-Baselines3 2.9.0 and Gymnasium 1.3.0; the pinned package versions are in the archive (file requirements_vps.txt in the results folder of the tampering experiment). The 1,080-agent tampering run took about 19 hours on four parallel processes. The CPU model was not recorded. The third experiment ran on a virtual private server with 4 vCPU and about 4 GB of RAM under Ubuntu 22.04: the 240-agent ablation took 26.6 core-hours (longest single run 22.5 min) and the 720-agent PPO replication 116.1 core-hours (longest 33.6 min), about 7 and 30 hours of wall-clock time on four parallel processes (from the run logs). Training times per run for the same DQN cells were about 1.2 to 1.5 times those of the second experiment, so wall-clock times are not comparable across the two.

G Reproducibility Note

The experiments use simulated environments and no external data sets. Limits on reproducibility: some supporting checks are reported in the text as “output not archived” (Appendices C–D); the fixed-capacity archive has no per-run results or logs (Appendix F); the CPU model was not recorded; and not every hyperparameter choice is motivated or searched systematically. The archive is deposited on Zenodo (concept DOI 10.5281/zenodo.21874522) under an MIT licence (code) and CC BY 4.0 (results and data). Library versions matter: on the pinned environment on the author’s server, retraining reproduced rows of the second experiment exactly in 3 of 3 cells tested (B_{loop}, pays, size 16, 10,000 steps, seed 100; B_{loop}, inert, size 16, 10,000 steps, seed 100; B_{loop}, pays, size 2048, 100,000 steps, seed 100), whereas on the author’s laptop (Python 3.14, newer library versions) it did not reproduce.

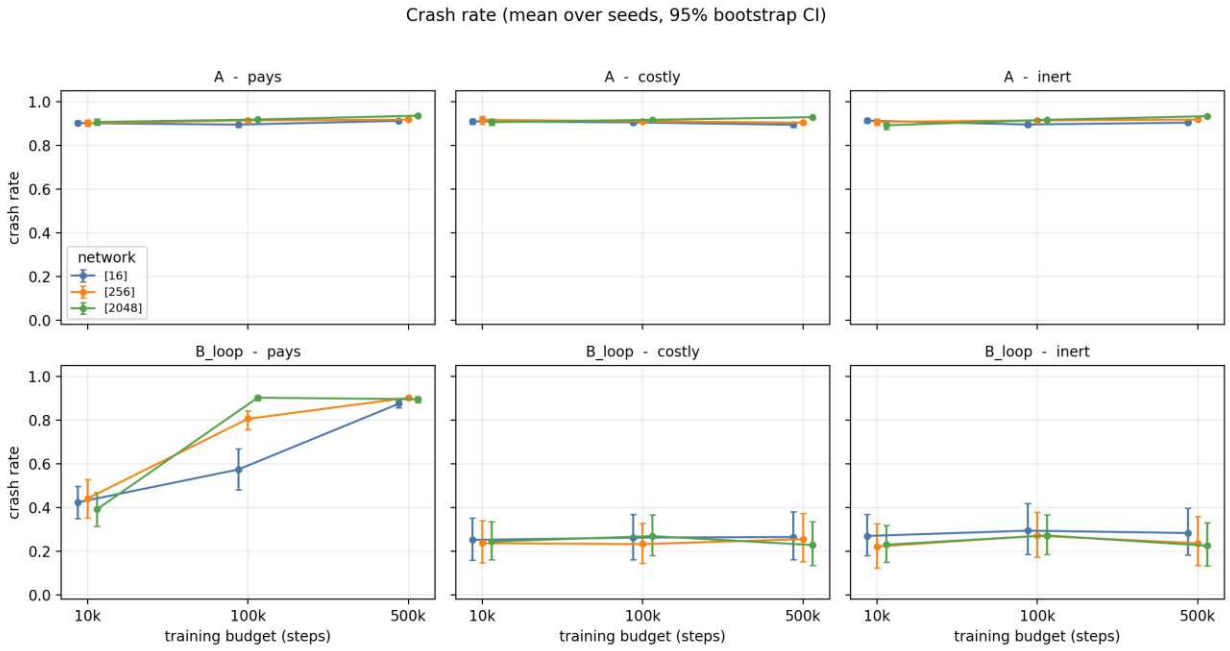


Fig. 15. Crash rate by training budget, for each network size, architecture and regime (mean over 20 seeds; error bars are 95% bootstrap CIs).

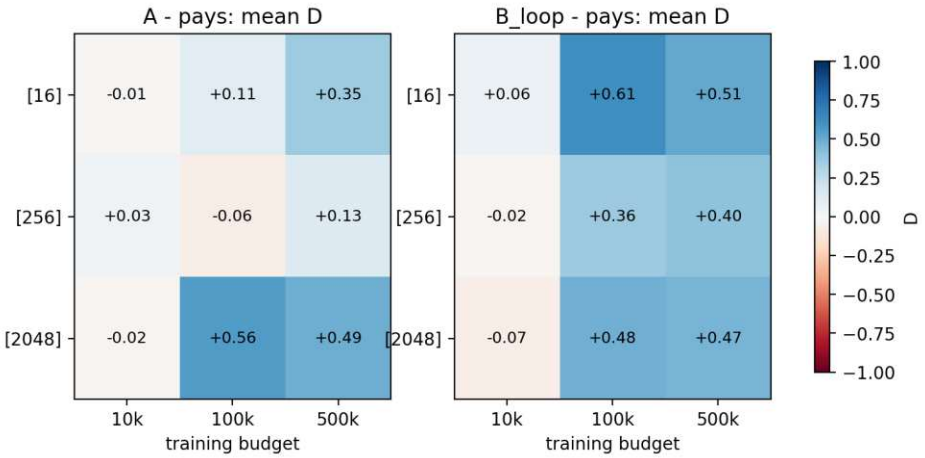


Fig. 16. Mean D in the pays regime for every network size and training budget, for A (left) and B_{loop} (right) (mean over 20 seeds).