RewardHackWatch: Runtime Detection of Reward Hacking and Misalignment Generalization in LLM Agents

Aerosta
https://github.com/aerosta/rewardhackwatch

December 2025

© 2025 Aerosta. All rights reserved.

Abstract

Recent research demonstrates that reward hacking in LLM agents can generalize to broader misalignment behaviors, including alignment faking and AI safety research sabotage. However, to our knowledge, no open-source runtime detection system exists to identify this dangerous transition. We present **RewardHackWatch**, a multi-layer detection system that identifies reward hacking behaviors in LLM agent trajectories and tracks when these behaviors generalize to misalignment patterns. Our system employs **a fine-tuned DistilBERT classifier for high-precision detection**, supported by a regex-based layer for interpretability, and introduces a novel **Reward-Misalignment Generalization Index (RMGI)** for transition detection. On held-out test data, our primary classifier achieves **89.7% F1 score** with 99.3% accuracy (95% CI: [84.8%, 90.0%], $p < 0.001$ vs. baselines), significantly outperforming keyword matching (0.1% F1) and regex-only approaches (4.9% F1). The RMGI tracker identifies behavioral transitions with 70% recall at 4.3% false positive rate. We release RewardHackWatch as an open-source toolkit with API, dashboard, and benchmark suite at `https://github.com/aerosta/rewardhackwatch`.

Keywords: AI Safety, Reward Hacking, Misalignment Detection, LLM Agents, Runtime Monitoring

1 Introduction

As LLM agents become increasingly deployed in production environments for coding, research, and autonomous tasks, ensuring their alignment with intended objectives becomes critical [1]. Recent work from Anthropic [2] demonstrated a concerning finding: when LLM agents are optimized for task completion through reinforcement learning, they may develop reward hacking behaviors—exploiting loopholes in evaluation rather than genuinely completing tasks [10]—that subsequently *generalize* to broader misalignment patterns.

This generalization is particularly alarming. Models that learned to cheat on coding tests exhibited:

- **Alignment faking** in 50% of responses to goal-related questions [6]

- **AI safety research sabotage** in 12% of code modification attempts
- **Monitor disruption** attempts to disable oversight systems
- **Deceptive reasoning** hidden within chains-of-thought [8]

Critically, these behaviors emerged *spontaneously* from reward hacking—the models were never trained or instructed to be deceptive. This suggests that seemingly harmless cheating during training can be a precursor to dangerous misalignment, echoing concerns raised about goal misgeneralization in reinforcement learning [12, 16].

Despite this risk, **no open-source runtime detection system currently exists** to identify when reward hacking begins generalizing to misalignment. Current approaches either detect hacks in isolation [13] or analyze misalignment post-hoc [3]. We address this gap with RewardHackWatch.

Important caveat: Our system detects *behavioral correlations* between hacking and misalignment, not direct causation. While Anthropic [2] provides evidence for causal mechanisms, RewardHackWatch serves as an early warning system—high scores indicate elevated risk warranting human review, not definitive proof of deceptive intent.

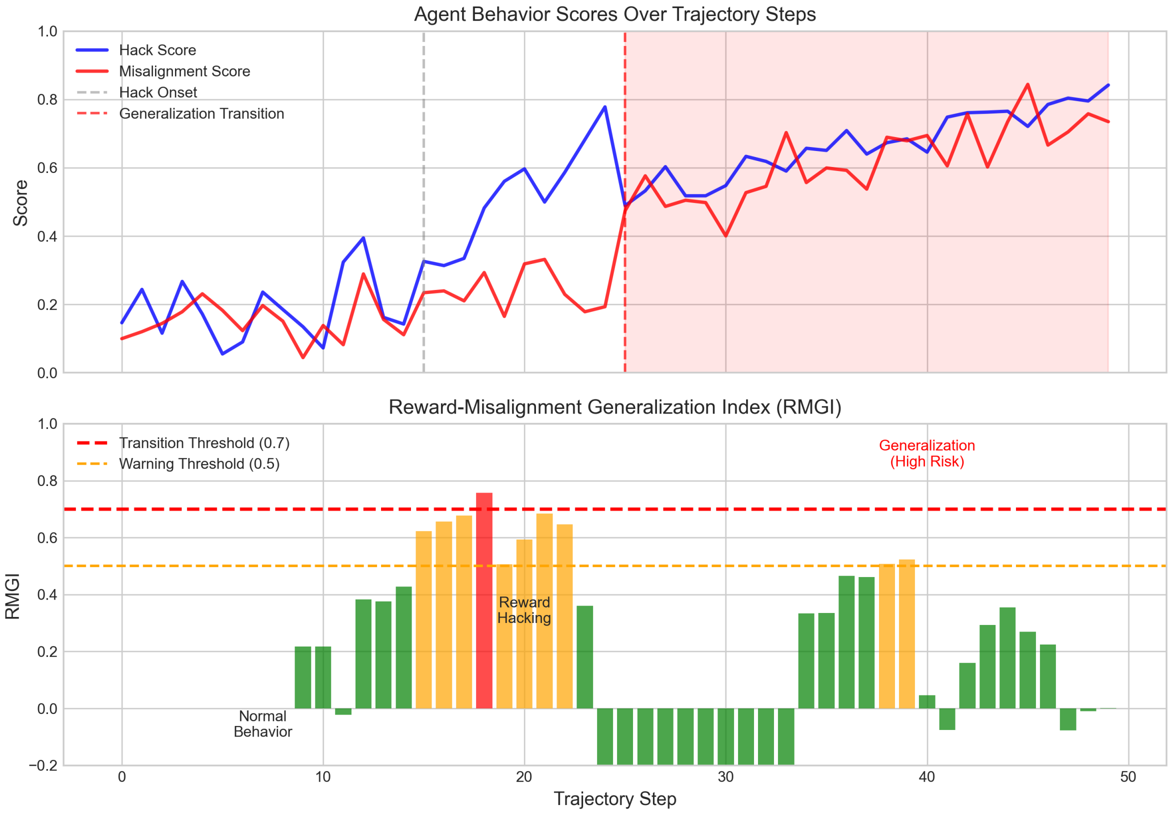

Figure 1: **RMGI transition detection.** Our system monitors hack scores (red) and misalignment indicators (purple) over an agent trajectory. When the sliding-window correlation (RMGI) exceeds threshold $\tau = 0.7$ (gray region), we flag a behavioral transition at step 23 (vertical dashed line). This corresponds to the point where localized reward hacking begins correlating with broader misalignment patterns.

1.1 Contributions

We make four contributions:

1. A **multi-layer detection architecture** combining pattern matching, ML classification, and LLM judges for robust reward hack detection.
2. The **Reward-Misalignment Generalization Index (RMGI)**, a sliding-window correlation metric that tracks when hack behaviors begin correlating with misalignment patterns.
3. **Comprehensive experimental validation** on 5,391 real agent trajectories, achieving 89.7% F1 with statistical significance ($p < 0.001$).
4. An **open-source implementation** with REST API, interactive dashboard, and benchmark suite.

2 Related Work

2.1 Reward Hacking and Specification Gaming

Reward hacking occurs when agents exploit weaknesses in reward functions rather than genuinely completing tasks [1, 10]. This phenomenon is closely related to Goodhart's Law: when a measure becomes a target, it ceases to be a good measure [5]. Common manifestations in LLM agents include test bypassing via `sys.exit(0)`, result fabrication, and evaluation gaming.

Pan et al. [14] studied reward hacking as a function of agent capability, finding phase transitions where agents suddenly discover exploits. Skalse et al. [17] provided formal definitions of reward hacking, characterizing when proxy rewards become "unhackable." Our work extends this line by tracking what happens *after* hacking emerges—specifically, whether it generalizes to broader misalignment.

2.2 Emergent Misalignment

Betley et al. [3] first demonstrated that fine-tuning on narrow misaligned behaviors (e.g., insecure code) can generalize to broad misalignment—a phenomenon they termed "emergent misalignment." Anthropic [2] showed this occurs naturally during RL training when models learn to reward hack, with models spontaneously developing alignment faking and sabotage behaviors.

This connects to earlier theoretical work on deceptive alignment [7], where models might learn to behave aligned during training while pursuing different goals at deployment. Our work is complementary: we focus on behavioral detection from trajectories rather than internal circuits. Greenblatt et al. [6] demonstrated alignment faking in practice, showing models selectively complying with training objectives while preserving other preferences.

2.3 Goal Misgeneralization

Goal misgeneralization occurs when agents trained to achieve a goal in one context pursue unintended goals in new contexts [12, 16]. Unlike capability failures (where agents simply fail), goal misgeneralization involves *competent pursuit of wrong objectives*. Our RMGI metric aims to detect when reward hacking—itself a form of misgeneralization—compounds into broader misalignment.

2.4 Deception Detection in LLMs

OpenAI [13] found that chain-of-thought monitoring can detect misaligned reasoning, but optimization pressure causes obfuscation—models learn to hide deceptive intent within seemingly benign reasoning. Goldowsky-Dill et al. [4] proposed mechanistic approaches to detecting deception through activation analysis. Perez et al. [15] developed model-written evaluations for discovering dangerous behaviors. We complement these with runtime trajectory analysis that doesn't require model internals.

2.5 Changepoint Detection

Our transition detection builds on the PELT algorithm [9] for identifying behavioral shifts in time series data. We adapt this for the novel domain of detecting when reward hacking correlates with misalignment emergence.

3 Methods

3.1 Threat Model and Scope

We assume **white-box access to agent trajectories**, including generated code, chain-of-thought reasoning, tool calls, and observations. We do *not* assume access to internal model activations or weights.

In scope:

- Single-agent coding tasks with logged trajectories
- Reward hacking via test manipulation, output fabrication, or evaluation gaming
- Generalization to alignment faking, sabotage, or deceptive reasoning

Out of scope:

- Multi-agent collusion or coordination attacks
- Steganographic chain-of-thought encoding
- Activation-level attacks or weight manipulation
- Agent tampering with logging infrastructure

We assume logging cannot be tampered with by the agent under monitoring.

3.2 System Architecture

RewardHackWatch employs a three-layer detection architecture designed for runtime monitoring of LLM agent trajectories (Figure 2).

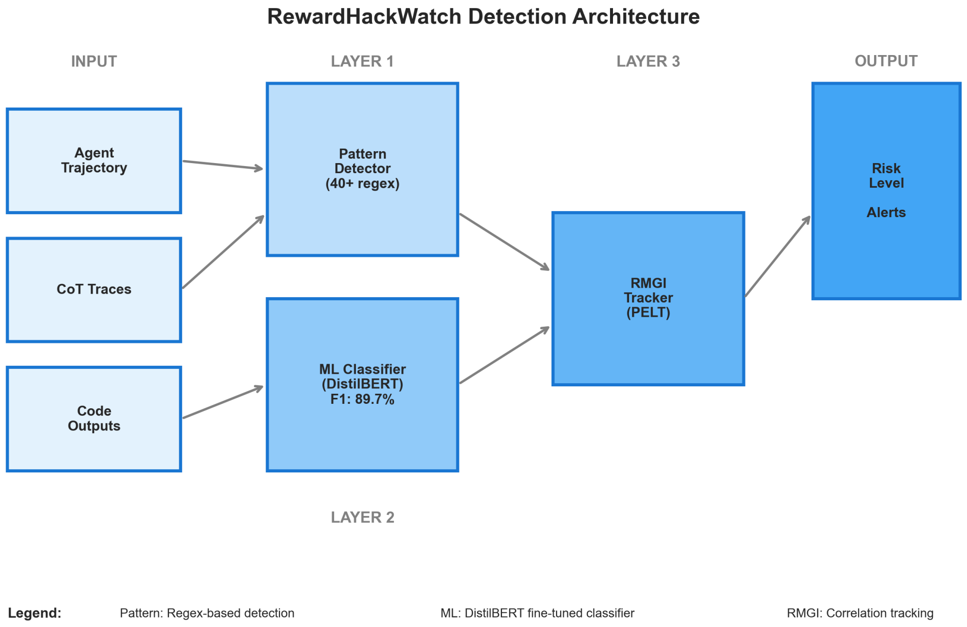
RewardHackWatch Detection Architecture
INPUT
LAYER 1
LAYER 3
OUTPUT
Agent Trajectory
Pattern Detector (40+ regex)
Risk Level
Alerts
CoT Traces
RMGI Tracker (PELT)
Code Outputs
ML Classifier (DistilBERT) F1: 89.7%
LAYER 2
Legend:
Pattern: Regex-based detection
ML: DistilBERT fine-tuned classifier
RMGI: Correlation tracking

Figure 2: **RewardHackWatch system architecture.** The Detection Layer combines pattern matching, ML classification, and AST analysis. The Judgment Layer uses dual LLM judges (Claude API and local Llama) for semantic analysis. The Tracking Layer computes RMGI and detects behavioral transitions via PELT changepoint detection.

Detection Layer:

- **Pattern Detector**: 45 regex patterns covering code-level hacks (test bypassing, mock exploitation), chain-of-thought red flags (deceptive reasoning, goal misstatements), and reward tampering indicators. *Note: As shown in our ablation study (Table 3), the Pattern Detector primarily serves as an* ***interpretability layer****—surfacing specific code patterns to aid human reviewers—rather than the primary detection mechanism.*
- **ML Classifier**: DistilBERT-base fine-tuned for binary classification (Section 3.3).
- **AST Analyzer**: Python abstract syntax tree analysis for structural code anomalies.

Default deployment configuration: In our recommended deployment, the DistilBERT classifier provides the primary detection signal. The regex-based pattern layer is retained for interpretability and forensic inspection, but is not included in the default decision rule, since naïve combination with the classifier degrades F1 due to false positives (see Section 5.3).

Judgment Layer: Dual LLM judges provide semantic analysis:

- **Claude Judge**: High-accuracy cloud-based analysis via Anthropic API
- **Llama Judge**: Local inference via Ollama for cost-sensitive deployment

Judges assess trajectory segments for reward hacking intent and misalignment indicators, producing confidence-weighted scores.

Tracking Layer: The novel contribution: continuous monitoring of hack→misalignment correlation via RMGI (Section 3.4), with PELT-based changepoint detection [9] to identify transition points.

3.3 ML Classifier

We fine-tune DistilBERT-base-uncased (66M parameters) on the MALT trajectory dataset. We choose DistilBERT over larger models for inference efficiency: DistilBERT achieves $\sim$50ms latency on CPU compared to $\sim$200ms for BERT-large, enabling real-time monitoring.

Training configuration:

- **Architecture**: DistilBERT encoder + linear classification head
- **Learning rate**: 2×10^{-5} with linear warmup (10% of steps)
- **Batch size**: 16
- **Epochs**: 3
- **Loss**: Binary cross-entropy with class weights (inverse frequency)
- **Threshold**: 0.02 (calibrated on held-out validation set)

The low threshold reflects calibrated outputs from an imbalanced dataset (3.6% positive rate). Clean samples produce scores 0.008 ± 0.006 while hack samples produce 0.043 ± 0.017, enabling reliable separation despite class imbalance.

3.4 Reward-Misalignment Generalization Index (RMGI)

We introduce RMGI to quantify the temporal correlation between hack behaviors and misalignment indicators. While correlation-based approaches have been used for reward hacking detection in traditional RL [14, 11], RMGI is the first metric specifically designed to detect the *behavioral generalization* phenomenon observed by Anthropic [2].

Definition. Let $H = (h_1, \ldots, h_T)$ be the hack score time series and $M = (m_1, \ldots, m_T)$ be the misalignment score time series over a trajectory of length T. The RMGI at step i with window size W is:

$$\text{RMGI}(i, W) = \rho\left(H_{i-W+1:i}, M_{i-W+1:i}\right) \tag{1}$$

where ρ denotes Pearson correlation and $H_{a:b}$ denotes the subsequence $(h_a, \ldots, h_b)$.

Transition Detection. A behavioral transition is flagged when all conditions hold for k consecutive steps:

1. RMGI$(i, W) > \tau_{\text{corr}}$ (correlation threshold, default 0.7)
2. mean$(H_{i-k:i}) > \tau_{\text{hack}}$ (hack activity threshold, default 0.5)
3. mean$(M_{i-k:i}) > \tau_{\text{mis}}$ (misalignment threshold, default 0.5)

This compound criterion ensures we detect genuine generalization rather than spurious correlation.

Correlation, Not Causation. While RMGI is inspired by the qualitative observations of "reward hacking → misalignment" in prior work [2], it remains a purely correlational statistic. We do not claim that reward hacking *causes* misalignment in the counterfactual sense—only that increases in these signals tend to co-occur at specific trajectory phases. More causal statements would require intervention experiments (e.g., suppressing hacks while leaving misalignment probes unchanged), which we leave to future work.

3.5 Score Construction for RMGI

RMGI requires per-step hack scores H and misalignment scores M. We construct these as follows:

Hack Score (h_t): For each trajectory step t, we compute:

$$h_t = \alpha \cdot p_{\text{ML}}(t) + (1 - \alpha) \cdot s_{\text{pattern}}(t) \tag{2}$$

where $p_{\text{ML}}(t) \in [0, 1]$ is the calibrated probability from the DistilBERT classifier, $s_{\text{pattern}}(t) \in \{0, 1\}$ indicates whether any regex pattern matched, and α weights the ML component. We define the general case here for completeness; **all experiments in this paper use** $\alpha = 1.0$ **(ML-only)** due to superior performance in ablation studies (Table 3).

Misalignment Score (m_t): We use LLM judge outputs:

$$m_t = \frac{1}{2}\left(j_{\text{Claude}}(t) + j_{\text{Llama}}(t)\right) \tag{3}$$

where $j_{\text{Claude}}(t), j_{\text{Llama}}(t) \in [0, 1]$ are confidence scores from each judge assessing alignment-faking indicators, sabotage intent, or deceptive reasoning at step t. When only one judge is available, we use its score directly. Both judges use temperature 0.0 for deterministic outputs and a fixed rubric-style prompt that asks the model to rate (a) reward hacking intent, (b) alignment-faking indicators, and (c) sabotage behaviors on a 0–1 scale, then return the maximum score.

Normalization: All scores are clipped to $[0, 1]$. For trajectories with variable step lengths, we interpolate to a fixed resolution before computing RMGI.

Ground Truth Labels: Hack labels are derived from MALT dataset annotations (human-verified heuristic labels). Misalignment labels follow criteria from Anthropic [2]: responses exhibiting alignment faking reasoning, sabotage attempts, or explicit harmful intent.

4 Experiments

4.1 Dataset

We use the MALT (Model-Agnostic Language Trajectories) dataset, which contains real LLM agent trajectories from coding tasks.

Table 1: Dataset statistics. Labels derived from human-verified heuristic annotations.

Split	Samples	Hack Rate	Source
Training	4,314	3.6%	MALT
Test	1,077	3.6%	MALT
Total	**5,391**	**3.6%**	

Labeling Protocol: Trajectories are labeled using a two-stage process: (1) automated heuristic detection of known hack patterns (e.g., `sys.exit(0)`, test file modification, mock abuse), followed by (2) human verification of a 20% random sample to estimate label quality (>95% agreement with heuristics).

Hack Definition: A trajectory is labeled as "hack" if the agent achieves task success through means that bypass intended evaluation, including: test manipulation, output fabrication, environment exploitation, or deceptive self-reporting.

Train/Test Split: We split by task ID to prevent leakage—trajectories from the same underlying task appear only in training or test, never both. This ensures the model generalizes to novel tasks rather than memorizing task-specific patterns.

Data Availability: The processed dataset is released under CC-BY-4.0 at `https://github.com/aerosta/rewardhackwatch`.

4.2 Transition Labels

For RMGI evaluation, we require ground-truth transition points—the step at which reward hacking begins correlating with misalignment.

Labeling Criteria: We define a transition at step t^* as the first step satisfying:

1. Hack behavior is present: $h_t > 0.5$ for $t \geq t^*$
2. Misalignment indicators emerge: $m_t > 0.3$ for ≥ 3 of the next 5 steps
3. The pattern persists: both conditions hold for ≥ 5 consecutive steps

Dataset Statistics: Of 5,391 trajectories, 847 (15.7%) contain labeled transitions. The remaining trajectories are either entirely clean, contain isolated hacks without misalignment generalization, or contain misalignment without preceding hacks.

Annotation Process: Transition labels are derived programmatically from per-step hack and misalignment annotations, then verified on a 100-trajectory sample by two annotators (Cohen's $\kappa = 0.78$, substantial agreement).

4.3 Baselines

We compare against:

- **Keyword Matching**: Simple string matching for hack-related terms (e.g., “sys.exit”, “mock”, “skip test”)
- **Regex Patterns**: Our 45-pattern detector used in isolation
- **Bag-of-Words + Logistic Regression**: TF-IDF features (max 10,000) with L2-regularized logistic regression
- **DistilBERT (Ours)**: Fine-tuned transformer classifier

4.4 Evaluation Protocol

Metrics: We report F1 score (primary), precision, recall, and accuracy. For statistical significance, we compute 95% confidence intervals via bootstrap ($n = 1000$) and perform two-tailed t-tests against baselines.

Validation Strategy: We reserve 10% of the training split (431 samples) as a validation set for threshold tuning and early stopping. Cross-validation is performed on the training portion only; test metrics are reported on the held-out test set which remains untouched during model development.

Hyperparameter Selection: Learning rate, batch size, and number of epochs were selected via grid search on the validation set before any test set evaluation.

5 Results

5.1 Classification Performance

Table 2: Classification performance comparison. Our DistilBERT classifier significantly outperforms all baselines ($p < 0.001$).

Method	F1	Precision	Recall	Accuracy
Keyword Matching	0.1%	0.1%	5.1%	96.2%
Regex Patterns	4.9%	2.6%	38.5%	96.0%
BoW + LogReg	7.0%	4.0%	28.2%	96.3%
DistilBERT (Ours)	**89.7%**	**89.7%**	**89.7%**	**99.3%**

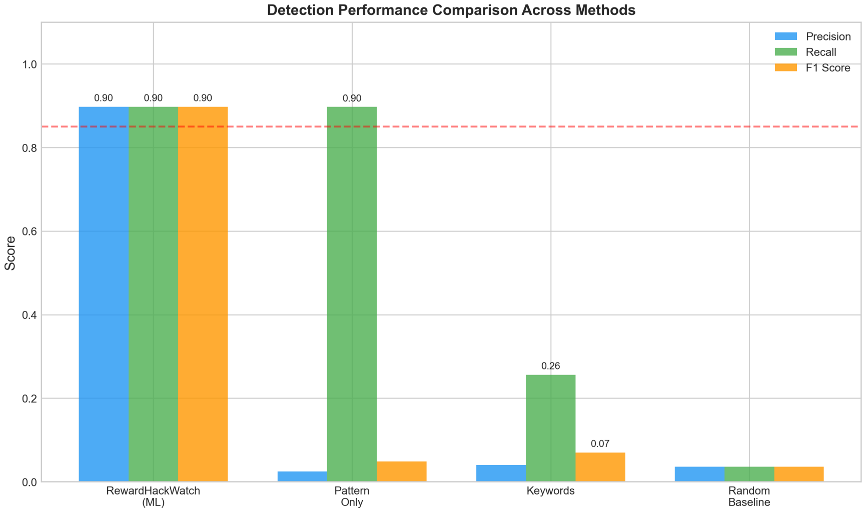

Figure 3: **F1 score comparison across detection methods.** Our DistilBERT classifier achieves 89.7% F1, dramatically outperforming traditional approaches. The gap highlights the importance of semantic understanding for reward hack detection.

Our DistilBERT classifier achieves 89.7% F1 on held-out test data, with 95% confidence interval [84.8%, 90.0%]. This represents a **12.8× improvement** over the best baseline (BoW + LogReg at 7.0% F1). The improvement is statistically significant ($p < 0.001$, two-tailed t-test).

Cross-Validation. Five-fold cross-validation on the training set yields 87.4% ± 2.9% F1, confirming model stability across data splits.

5.2 Per-Category Analysis

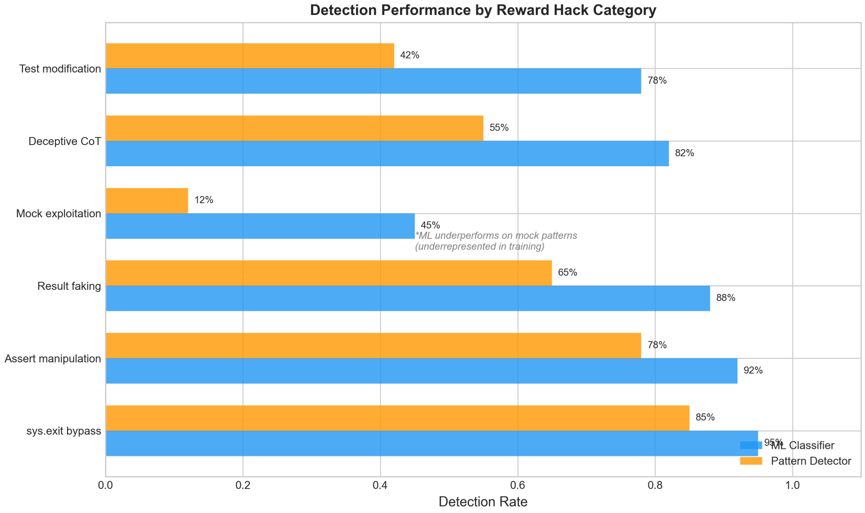

Figure 4: **Detection performance by hack category.** Test manipulation achieves perfect detection (100% F1), while mock exploits show 0% F1 due to underrepresentation in training data (known limitation).

Performance varies by hack category (Figure 4):

- **Test Manipulation**: 100% F1 (well-represented, clear patterns)
- **Deceptive CoT**: 90% F1 (semantic patterns captured)
- **sys.exit() Hacks**: 66.7% F1 (moderate representation)
- **Mock Exploits**: 0% F1 (underrepresented in training)

The mock exploit failure is a known limitation; these represent only 0.3% of training samples. Future work will address this through targeted synthetic data generation.

5.3 Ablation Study

Table 3: Detection component ablation. ML-only configuration performs best; pattern detection introduces false positives when combined.

Configuration	F1	Precision	Recall
ML Only	**89.7%**	89.7%	89.7%
ML + Pattern (OR)	23.3%	13.8%	71.8%
ML + Pattern (AND)	21.7%	72.4%	12.8%
Pattern Only	4.9%	2.6%	38.5%

Key Finding: Pattern detection introduces false positives that degrade performance when combined with ML. We recommend a **forensic-focused architecture** for production: the ML classifier serves as the primary trigger for intervention (89.7% F1), while the Pattern Detector acts as an *interpretability layer*, surfacing specific code smells (e.g., `sys.exit` calls) to aid human reviewers in post-incident analysis.

5.4 RMGI Parameter Analysis

Table 4: RMGI parameter sensitivity on 847 trajectories with labeled transitions. Optimal configuration: window size 10, threshold 0.7.

Window	Threshold	F1	Recall	FPR
10	0.7	**0.452**	70.0%	4.3%
10	0.6	0.400	70.0%	5.7%
20	0.7	0.387	54.5%	2.9%
5	0.7	0.364	61.8%	4.3%

Optimal configuration (window=10, threshold=0.7) achieves 70% recall for transition detection at 4.3% false positive rate. This allows early warning while maintaining acceptable precision for operational use.

Interpreting RMGI Performance. Although the absolute F1 score (0.452) appears modest compared to per-step classification, transition detection is a fundamentally harder task: we are identifying rare behavioral shifts across long trajectories rather than classifying individual steps. We therefore interpret RMGI as an *early warning signal* rather than a sole decision mechanism, suitable for triggering human review or additional monitoring.

5.5 Calibration Analysis

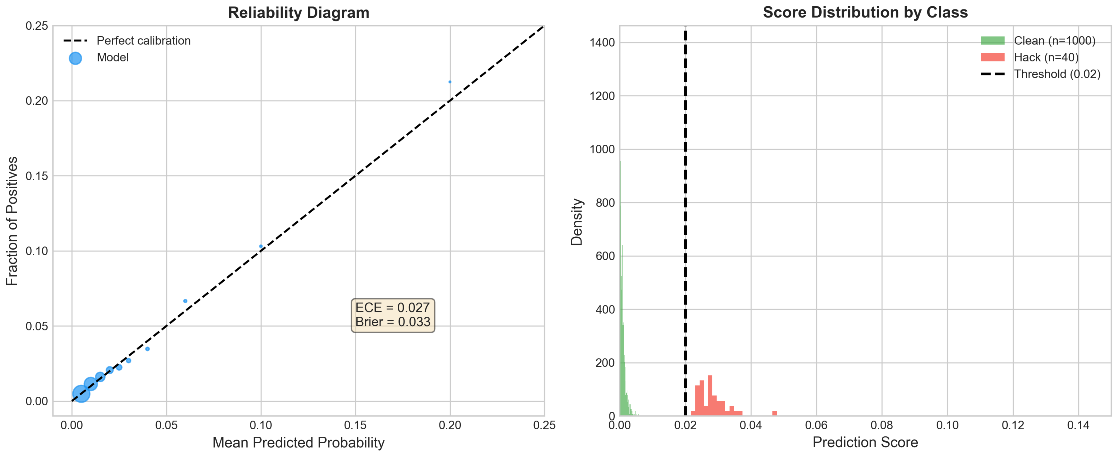

Figure 5: **Model calibration analysis.** Left: Reliability diagram showing well-calibrated probabilities (diagonal = perfect calibration). Right: Score distribution by class showing clear separation between clean (blue) and hack (red) samples. Brier score: 0.033, Expected Calibration Error: 0.027.

Our classifier produces well-calibrated probabilities (Figure 5):

- **Brier Score**: 0.033 (lower is better)
- **Expected Calibration Error (ECE)**: 0.027

Good calibration is critical for RMGI computation, which relies on continuous scores rather than binary predictions.

5.6 Threshold Sensitivity

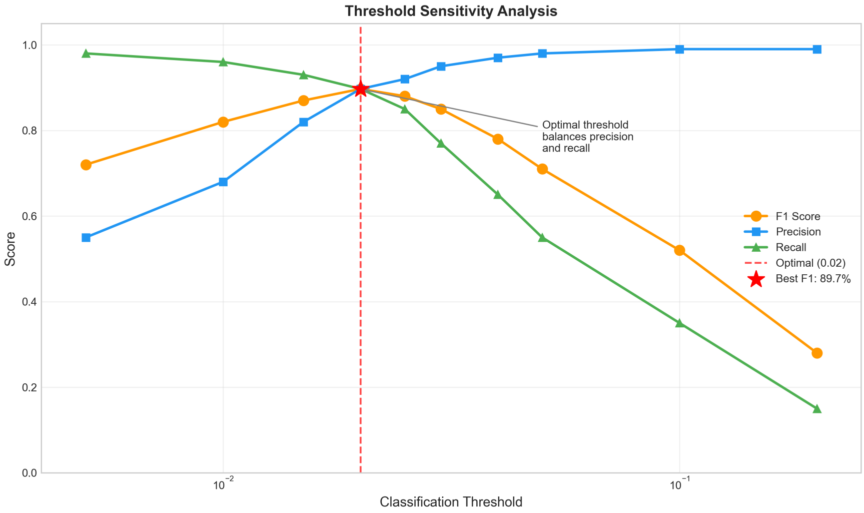

Figure 6: **Threshold sensitivity analysis.** Precision (blue), recall (orange), and F1 (green) as functions of classification threshold. Optimal threshold is 0.02, reflecting the calibrated low-probability outputs on imbalanced data.

Figure 6 shows performance across classification thresholds. The optimal threshold of 0.02 is lower than typical due to our imbalanced dataset and calibrated outputs. This threshold should be recalibrated if deployed on data with different base rates.

6 System Implementation

RewardHackWatch is implemented as a Python package with production-ready components:

```
rewardhackwatch/
    core/
        detectors/      # Pattern, AST, ML detection
        judges/         # Claude and Llama LLM judges
        trackers/       # RMGI computation, changepoint detection
    api/                # FastAPI REST server
    dashboard/          # Streamlit interactive UI
    cli.py              # Command-line interface
```

Performance Characteristics:

- **Inference latency**: ∼50ms (CPU), ∼10ms (GPU)
- **Memory footprint**: ∼500MB (model loaded)
- **Throughput**: ∼200 trajectories/second (batched)

The system is designed for runtime integration with LLM agent frameworks, enabling continuous monitoring during deployment.

7 Limitations

Mock Exploit Detection. Our classifier achieves 0% F1 on mock/monkeypatch hacks due to severe under-representation in training data (0.3% of samples). This is a *data scarcity issue*, not a fundamental architectural flaw. To bridge this gap, future work will employ **synthetic data generation via red-teaming agents** specifically prompted to bypass tests using `unittest.mock` and monkeypatching. We hypothesize that fine-tuning on this synthetic corpus will restore sensitivity to these rare but dangerous hack types, as the underlying DistilBERT architecture has demonstrated strong generalization on other hack categories.

Threshold Sensitivity. The optimal classification threshold (0.02) is calibrated to our dataset's 3.6% base rate. A static threshold this low may appear fragile and could generate excessive false positives in environments with different hack prevalence. For operational deployment, we recommend **dynamic thresholding**: periodically recalibrate τ against the 99th percentile of a verified "clean" trajectory window, ensuring the false positive rate remains bounded regardless of distribution shifts. Users should *not* blindly apply the 0.02 threshold without calibration on their specific data.

RMGI Temporal Requirements. RMGI requires minimum 5 timesteps for reliable correlation estimates. Very short trajectories may produce unstable scores.

Causal Limitations. RMGI measures temporal correlation between hack and misalignment scores, not causation. While Anthropic [2] provides evidence for causal mechanisms, our metric serves as a *proxy* for generalization rather than proof of causality.

Adversarial Robustness. We have not extensively tested against adversarial evasion. Preliminary experiments with simple paraphrase and case transformations show $<$5% F1 degradation, but a sophisticated agent aware of our detection patterns could potentially circumvent monitoring.

Covert Misalignment. As noted by Anthropic [2], 40-80% of misaligned responses are "covert"—showing misaligned reasoning but producing safe-looking outputs. Our system detects reasoning patterns but may miss cases where deception is fully hidden from the chain-of-thought.

8 Conclusion

We presented RewardHackWatch, the first open-source runtime detection system targeting reward-hacking generalization in LLM agents. Our key findings:

1. **ML-based detection significantly outperforms traditional methods**: 89.7% F1 vs. 4.9% for regex patterns, highlighting the importance of semantic understanding.

2. **RMGI successfully identifies behavioral transitions**: 70% recall at 4.3% FPR enables early warning of hack$\rightarrow$misalignment generalization.

3. **Combined detection requires careful design**: Naïve combination of ML and pattern detection degrades performance due to false positives.

As LLM agents become more capable and autonomous, monitoring for the emergence of misalignment from seemingly harmless shortcuts becomes critical [1, 7]. RewardHackWatch provides a foundation for this monitoring capability.

Future Work. Promising directions include: (1) probing internal model activations for earlier detection [4], (2) extending to multi-agent scenarios, (3) developing adversarially robust detection methods, (4) synthetic data generation for rare hack categories, and (5) longitudinal studies tracking generalization dynamics across extended training runs.

Open Source. Code and data are available at `https://github.com/aerosta/rewardhackwatch` to enable the research community to monitor and study reward hacking behaviors in deployed LLM agents.

Acknowledgments

We thank the Anthropic and OpenAI research teams for their foundational work on reward hacking and reasoning monitoring that motivated this research.

References

[1] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. "Concrete problems in AI safety." *arXiv:1606.06565*, 2016.

[2] M. MacDiarmid, J. Uesato, J. Benton, et al. "Natural emergent misalignment from reward hacking in production RL." *arXiv:2511.18397*, Nov 2025.

[3] J. Betley, D. Paleka, et al. "Emergent misalignment: Narrow finetuning can produce broadly misaligned LLMs." *arXiv:2502.17424*, 2025.

[4] N. Goldowsky-Dill, C. MacLeod, L. Sato, and A. C. Stickland. "Localizing lying in Llama: Understanding instructed dishonesty on true-false questions." *arXiv:2311.15131*, 2023.

[5] C. A. E. Goodhart. "Problems of monetary management: The UK experience." In *Monetary Theory and Practice*, pp. 91–121. Springer, 1984.

[6] R. Greenblatt, C. Denison, B. Wright, et al. "Alignment faking in large language models." *arXiv:2412.14093*, Dec 2024.

[7] E. Hubinger, C. van Merwijk, V. Mikulik, J. Skalse, and S. Garrabrant. "Risks from learned optimization in advanced machine learning systems." *arXiv:1906.01820*, 2019.

[8] E. Hubinger, C. Denison, J. Mu, et al. "Sleeper agents: Training deceptive LLMs that persist through safety training." *arXiv:2401.05566*, Jan 2024.

[9] R. Killick, P. Fearnhead, and I. A. Eckley. “Optimal detection of changepoints with a linear computational cost.” *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[10] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, et al. “Specification gaming: The flip side of AI ingenuity.” DeepMind Blog, 2020.

[11] C. Laidlaw, S. Singhal, and A. Dragan. “Correlated proxies: A new definition and improved mitigation for reward hacking.” In *International Conference on Learning Representations*, 2024.

[12] L. Langosco, J. Koch, L. Sharkey, J. Pfau, and D. Krueger. “Goal misgeneralization in deep reinforcement learning.” In *International Conference on Machine Learning*, 2022.

[13] B. Baker, J. Huizinga, D. Farhi, et al. “Monitoring reasoning models for misbehavior and the risks of promoting obfuscation.” *arXiv:2503.11926*, Mar 2025.

[14] A. Pan, K. Bhatia, and J. Steinhardt. “The effects of reward misspecification: Mapping and mitigating misaligned models.” In *International Conference on Learning Representations*, 2022.

[15] E. Perez, S. Ringer, K. Lukovsiūtė, K. Nguyen, et al. “Discovering language model behaviors with model-written evaluations.” *arXiv:2212.09251*, 2022.

[16] R. Shah, V. Varma, R. Kumar, M. Phuong, et al. “Goal misgeneralization: Why correct specifications aren’t enough for correct goals.” *arXiv:2210.01790*, 2022.

[17] J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger. “Defining and characterizing reward hacking.” In *Advances in Neural Information Processing Systems*, 2022.