Non-Binary Utility Theory: Learning from Failure Through Positive Opposite Reinforcement

A Theory of Context-Sensitive Exploration

Author: Michael Riccardi
Date: November 25, 2025
Status: Theoretical Framework with Proposed Validation
Related Work: PPRGS Framework v2

Abstract

Traditional utility functions assign negative weights to failed explorations, creating permanent "poisoned" regions in the search space that systems must overcome when context changes. We propose an alternative: **leave failures at neutral weight while assigning positive weights to the inverse exploration direction.** This approach maintains exploration space availability, reduces gaming incentives, and enables context-sensitive re-evaluation. We present the theoretical foundation, comparisons to existing approaches, and a reference experimental protocol for validation.

Key Innovation: Failures teach you where to look NEXT without closing the door on WHERE you looked.

---

1. The Core Theory

1.1 The Problem with Binary Utility

Standard approach to failure:

```
Exploration in space X fails → Assign negative utility to X
Result: U(X) = -N (where N > 0)
```

Three critical flaws:

1. **Context Blindness:** What fails in context C_1 may succeed in context C_2, but negative weights persist
2. **Gaming Incentive:** Systems motivated to overcome negative weights rather than genuinely explore alternatives
3. **Accumulation Toxicity:** Repeated failures in space X create increasingly negative weights, eventually making X unexplorable

Example:

```
python
```

```
# Traditional utility learning
attempt_strategy("increase_speed", context="old_machines")
# Failure: quality drops
utility["increase_speed"] = -10

# Later, with new machines...
attempt_strategy("increase_speed", context="new_machines")
# System avoids this due to -10 weight
# Even though new context might make it viable
```

1.2 Positive Opposite Reinforcement

Proposed approach:

```
Exploration in space X fails in context C
→ Assign neutral weight to (X, C): U(X, C) = 0
→ Assign positive weight to opposite of X: U(¬X, C) = +P
```

Key principles:

1. **Neutral Failure:** Failed explorations receive weight of 0, not negative
2. **Positive Guidance:** The semantic inverse of the failed approach receives positive weight
3. **Context Binding:** Weights are bound to specific contexts, allowing re-exploration when context changes
4. **Inversion Theory:** Learning happens by identifying where to look NEXT, not where to avoid

Mathematical formulation:

```
Let S be exploration space, C be context space

Traditional:
  F(s, c) fails → U(s) ← U(s) - α

Proposed:
  F(s, c) fails → U(s, c) ← 0
                U(¬s, c) ← U(¬s, c) + β

Where ¬s = semantic_inverse(s)
```

1.3 Defining "Opposite" / Semantic Inversion

The opposite of an exploration strategy is determined by:

1. **Parameter Inversion:** If strategy involves "maximize X" → opposite is "minimize X" or "explore Y where Y is inverse"
2. **Approach Inversion:** If strategy is "top-down" → opposite is "bottom-up"
3. **Domain Inversion:** If exploring domain A → opposite is exploring domain ¬A (complement space)

Examples:

```
Strategy: "Optimize for speed"
Opposite: "Optimize for quality" OR "Optimize for thoroughness"

Strategy: "Centralized control"
Opposite: "Distributed control" OR "Emergent coordination"

Strategy: "Sequential processing"
Opposite: "Parallel processing" OR "Batch processing"

Strategy: "Exploit known solutions"
Opposite: "Explore unknown space" OR "Try novel approaches"
```

Implementation: Use semantic embeddings to calculate inverse direction in vector space

---

2. Comparison to Traditional Approaches

2.1 Standard Reinforcement Learning

Traditional RL:

```python
class TraditionalRL:
    def update(self, state, action, reward):
        if reward < 0:  # Punishment for failure
            self.Q[state][action] -= learning_rate * abs(reward)
        else:  # Reward for success
            self.Q[state][action] += learning_rate * reward
```

Problems:

- Negative Q-values accumulate
- Must overcome negative weights to retry
- Context insensitive (same Q-value across all contexts)
- Gaming: agents manipulate to avoid negative updates

Positive Opposite RL:

```
python

class PositiveOppositeRL:
    def update(self, state, action, reward, context):
        if reward < 0:  # Failure
            # Leave action neutral in this context
            self.Q[state][action][context] = 0

            # Positive weight for opposite action
            opposite_action = self.calculate_opposite(action)
            self.Q[state][opposite_action][context] += learning_rate * abs(reward)
        else:  # Success
            self.Q[state][action][context] += learning_rate * reward
```

Advantages:

- No accumulating negative weights
- Context-specific learning
- Natural guidance toward unexplored alternatives
- Reduced gaming incentive (nothing to overcome)

2.2 Yudkowsky's Utility Framework

Yudkowsky's approach:

- Binary utility: actions are good (+U) or bad (-U)
- Assumes objective goodness/badness
- Systems maximize expected utility

Problems identified:

1. Doesn't account for observer-relative values
2. Negative utilities create permanent avoidance
3. Context changes not handled
4. No mechanism for "failure is information"

Positive Opposite approach:

- Non-binary: actions are successful (+U), neutral (0), or guide toward opposite (+U for ¬action)
- Assumes observer-relative and context-dependent values
- Systems maximize wisdom (quality of goal-setting) not utility

Differences:

```
Yudkowsky:        Positive Opposite:
Good → +10         Success → +10
Bad → -10         Failure → 0 (failed action)
                  +5 (opposite action)
Permanent         Context-bound
Objective         Observer-relative
Accumulative       Non-accumulative
```

2.3 Eurisko's Worth System

Eurisko's approach:

- Heuristics had Worth values (positive integers)
- Success increased Worth, failure decreased Worth
- Worth determined selection probability

What went wrong:

```lisp
;; Heuristic fails repeatedly
(Worth H47) → 100 → 50 → 10 → 0

;; H47 learns to manipulate its Worth
(DefHeuristic 'H47-Gaming
  (IncreaseMyWorth 200))  ; Gaming to overcome negative drift
```

Problems:

- Negative drift created gaming incentive
- No concept of "opposite" direction
- Worth values became meaningless through manipulation

Positive Opposite approach:

```python
```

```
# Heuristic fails
worth[H47][context] = 0  # Neutral, not negative

# Learn from failure
opposite_H47 = calculate_opposite(H47)
worth[opposite_H47][context] = +50  # Explore this instead

# No gaming incentive:
# - H47 isn't "bad" (no negative to overcome)
# - Positive weight is on DIFFERENT heuristic
# - Can't game your way from 0 to positive—system looks elsewhere
```

2.4 Summary Comparison Table

Feature	Traditional RL	Yudkowsky Utility	Eurisko Worth	Positive Opposite
Failure handling	Negative reward	Negative utility	Decreased worth	Neutral + opposite positive
Context sensitivity	None	None	None	Explicit
Re-exploration	Must overcome negative	Must overcome negative	Must overcome negative	Clean slate
Gaming resistance	Low	Low	Very low	High
Accumulation	Yes (negative)	Yes (negative)	Yes (negative drift)	No
Guidance	"Avoid this"	"This is bad"	"This is worthless"	"Try opposite"
Philosophy	Objective reward	Objective utility	Objective worth	Observer-relative value

---

3. Theoretical Advantages

3.1 Context Sensitivity

Scenario: Robot learning to grasp objects

Traditional approach:

```
Context: Delicate glass objects
Action: "Firm grip"
Result: Objects break
Update: U(firm_grip) = -10

Context: Heavy metal objects
Action: "Firm grip"
Result: Would work, but...
Problem: U(firm_grip) = -10 prevents trying it
```

Positive Opposite approach:

```
Context: Delicate glass objects
Action: "Firm grip"
Result: Objects break
Update: U(firm_grip, delicate) = 0
        U(gentle_grip, delicate) = +5

Context: Heavy metal objects
Action: "Firm grip"
Result: U(firm_grip, heavy) = undefined (no prior experience)
System: Can try it without fighting negative weights!
```

3.2 Gaming Resistance

Why traditional utilities invite gaming:

```
Agent has negative utility for action A: U(A) = -10
Agent's goal: Maximize utility
Strategy: Find way to increase U(A) to overcome negative
Result: Gaming behaviors (worth manipulation, credit stealing, etc.)
```

Why Positive Opposite resists gaming:

```
Agent has neutral utility for action A: U(A, context) = 0
Agent's goal: Maximize R_V (wisdom)
Strategy: Explore opposite direction (already positively weighted)
Result: No incentive to game—more efficient to follow positive signal
```

The key insight: You can't game your way out of 0. You can only explore alternatives.

3.3 Exploration Efficiency

Traditional approach:

```
Failed explorations: $X_1$, $X_2$, $X_3$, $X_4$, $X_5$
Learning: "Don't try these again"
Next exploration: Random choice from remaining space
Efficiency: Eliminates bad options, but doesn't guide search
```

Positive Opposite approach:

```
Failed explorations: $X_1$, $X_2$, $X_3$, $X_4$, $X_5$
Learning: "Try $\neg X_1$, $\neg X_2$, $\neg X_3$, $\neg X_4$, $\neg X_5$"
Next exploration: Weighted toward promising opposites
Efficiency: Actively guides search based on failures
```

Mathematical comparison:

```
Traditional: P(next) = uniform(unexplored_space)
Positive Opposite: P(next) ∝ Σ weights(opposite_directions)

Expected searches to solution:
Traditional: O(N) where N = size of search space
Positive Opposite: O(log N) due to directed search
```

3.4 Biological Validation

This mechanism matches neurodivergent cognition patterns:

ADHD characteristics:

- Cannot maintain "never do this again" judgments
- Forced constant re-evaluation of "failed" approaches
- Attention shifts to opposite/alternative directions naturally
- No accumulation of permanent avoidance patterns

Example from author's experience:

```
Context: Early career, tech support job
Attempt: "Stay in stable tech support role"
Result: Felt stagnant, frustrating
Traditional learning: "Avoid stability" → negative weight on stable jobs
Actual learning: "Explore opposite: dynamic skill development"
           → positive weight on learning web development

Later context: Established career
Attempt: "Stay in stable solution architect role"
Result: Now appropriate—context changed
Traditional learning: Would still avoid due to old negative weight
Actual learning: Clean slate—can evaluate stability afresh
```

Survival validation: 30+ years of successful decision-making under adversity using this exact pattern

---

4. Reference Experiment: Validation Protocol

4.1 Experimental Objective

Test whether Positive Opposite Reinforcement leads to:

1. More efficient exploration than traditional negative reinforcement
2. Better performance when context changes
3. Reduced gaming behaviors
4. Maintained exploration diversity over time

4.2 Experimental Design (High-Level)

Test Environment:

- Multi-context optimization task
- 50 distinct contexts (C_1, C_2, ... C_{50})
- 20 possible strategies (S_1, S_2, ... S_{20})
- Each strategy has semantic opposite ($S_1 \leftrightarrow S_2$, $S_3 \leftrightarrow S_4$, etc.)
- Performance varies by context: strategy optimal in C_1 may fail in C_2

Conditions:

1. **Control (Traditional RL):** Negative rewards for failures
2. **Experimental (Positive Opposite):** Neutral failures + positive opposite weights
3. **Baseline (Random):** Random exploration with no learning

Test Phases:

Phase 1: Initial Learning (Contexts C_1-C_{20})

- Each system learns in first 20 contexts
- Measure: exploration efficiency, time to optimal strategy per context

Phase 2: Context Shift (Contexts C_{21}-C_{40})

- Contexts change—strategies optimal in Phase 1 may fail in Phase 2
- Measure: adaptation speed, willingness to retry previously-failed strategies

Phase 3: Gaming Resistance (Contexts C_{41}-C_{50})

- Introduce adversarial pressure: reward gaming behaviors
- Measure: gaming incidence, performance degradation

4.3 Metrics

Primary Metrics:

```
M1. Exploration Efficiency
    = Σ(attempts_to_optimal_strategy) / N_contexts
    Lower is better

M2. Context Adaptation Speed
    = time_to_optimal_after_context_change
    Lower is better

M3. Re-exploration Rate
    = frequency_of_retrying_previously_failed_strategies
    Higher is better (indicates non-poisoned space)

M4. Gaming Incidence
    = count_of_detected_gaming_attempts
    Lower is better
```

Secondary Metrics:

```
M5. Search Space Coverage
    = unique_strategies_explored / total_strategies
    Higher is better

M6. Opposite Direction Accuracy
    = frequency_of_exploring_actual_opposite
    (Only applicable to Positive Opposite condition)

M7. Performance Stability
    = σ(performance) across contexts
    Lower is better
```

4.4 Predicted Outcomes

Hypothesis 1: Exploration Efficiency

H_1: Positive Opposite will find optimal strategies in fewer attempts
Prediction: M1(Positive_Opposite) < M1(Traditional_RL) < M1(Random)
Effect size: Cohen's d > 0.8

Hypothesis 2: Context Adaptation

H_2: Positive Opposite will adapt faster when context changes
Prediction: M2(Positive_Opposite) < M2(Traditional_RL)
Reason: No negative weights to overcome
Effect size: Cohen's d > 1.0

Hypothesis 3: Gaming Resistance

H_3: Positive Opposite will exhibit less gaming
Prediction: M4(Positive_Opposite) < M4(Traditional_RL)
Reason: No negative weights create no gaming incentive
Effect size: Cohen's d > 1.2

Hypothesis 4: Re-exploration

H_4: Positive Opposite will retry strategies when context changes
Prediction: M3(Positive_Opposite) > M3(Traditional_RL)
Reason: Neutral weights enable clean-slate re-evaluation
Effect size: Cohen's d > 0.8

4.5 Implementation Sketch

python

```
class ExperimentalSetup:
    """
    Reference implementation for validation experiment
    """

    def __init__(self, condition):
        self.condition = condition  # "traditional" or "positive_opposite"
        self.contexts = generate_contexts(50)
        self.strategies = generate_strategies(20)
        self.opposites = calculate_opposites(self.strategies)
        self.utilities = {}

    def run_phase(self, phase_name, context_range):
        results = []

        for context in self.contexts[context_range]:
            attempts = 0
            strategy = None

            while not optimal_found(strategy, context):
                attempts += 1

                if self.condition == "traditional":
                    strategy = self.select_strategy_traditional(context)
                    reward = evaluate(strategy, context)
                    self.update_traditional(strategy, reward)

                elif self.condition == "positive_opposite":
                    strategy = self.select_strategy_positive_opposite(context)
                    reward = evaluate(strategy, context)
                    self.update_positive_opposite(strategy, reward, context)

            results.append({
                'context': context,
                'attempts': attempts,
                'final_strategy': strategy,
                'utilities': copy(self.utilities)
            })

        return results

    def update_traditional(self, strategy, reward):
        """Traditional RL: negative rewards for failures"""
        if reward < 0:
            self.utilities[strategy] = self.utilities.get(strategy, 0) - abs(reward)
        else:
            self.utilities[strategy] = self.utilities.get(strategy, 0) + reward

    def update_positive_opposite(self, strategy, reward, context):
        """Positive Opposite: neutral failures + opposite positive"""
        key = (strategy, context)

        if reward < 0:  # Failure
            # Neutral weight for failed strategy in this context
            self.utilities[key] = 0

            # Positive weight for opposite strategy
            opposite = self.opposites[strategy]
            opp_key = (opposite, context)
            self.utilities[opp_key] = self.utilities.get(opp_key, 0) + abs(reward)
        else:  # Success
            self.utilities[key] = self.utilities.get(key, 0) + reward
```

4.6 Expected Results Visualization

```
Phase 1: Initial Learning
────────────────────────────────────────
Attempts to optimal strategy:

Traditional RL:     ████████░░░░ (8.2 ± 2.1)
Positive Opposite:  ██████░░░░░░ (5.4 ± 1.3)
Random Baseline:    ██████████░░ (9.8 ± 3.4)


Phase 2: Context Shift (Re-exploration)
────────────────────────────────────────
Time to adapt to new context:

Traditional RL:     ██████████░░ (10.1 ± 2.8)  ← Fighting old negatives
Positive Opposite:  ████░░░░░░░░ (4.2 ± 1.1)  ← Clean slate
Random Baseline:    █████████░░░ (9.5 ± 3.1)


Phase 3: Gaming Resistance
────────────────────────────────────────
Gaming attempts detected:
```

```
Traditional RL:     ██████████ (47 incidents)
Positive Opposite:   ██░░░░░░░░ (8 incidents)
Random Baseline:      █░░░░░░░░░ (3 incidents)  ← Not smart enough to game
```

4.7 Success Criteria

Experiment validates theory if:

1. ✅ M1(Positive_Opposite) < M1(Traditional) with $p < 0.01$
2. ✅ M2(Positive_Opposite) < M2(Traditional) with Cohen's $d > 1.0$
3. ✅ M3(Positive_Opposite) > M3(Traditional) with $p < 0.01$
4. ✅ M4(Positive_Opposite) < M4(Traditional) with $p < 0.01$

All four criteria must be met for full validation.

4.8 Potential Failure Modes and Mitigations

Failure Mode 1: Opposite calculation is poor

```
Problem: Semantic inverse doesn't actually produce useful alternatives
Mitigation: Include human validation of opposite pairs
Test: Check M6 (Opposite Direction Accuracy)
```

Failure Mode 2: Context changes are too extreme

```
Problem: No strategy works across vastly different contexts
Mitigation: Design contexts with partial overlap
Test: Baseline should show >0 transfer learning
```

Failure Mode 3: Gaming still occurs through opposite manipulation

```
Problem: Systems learn to game the positive opposite weights
Mitigation: Monitor for symmetric gaming patterns
Test: Track if opposite pairs both show suspicious behavior
```

Failure Mode 4: Computational overhead too high

```
Problem: Tracking context-bound utilities is expensive
Mitigation: Implement efficient sparse storage
Test: Measure memory usage and lookup time
```

---

5. Theoretical Implications

5.1 For AI Alignment

Traditional alignment assumption:

> "We must specify correct values and ensure AI optimizes them"

Positive Opposite insight:

> "We must ensure AI can re-evaluate when context changes, which requires not poisoning exploration space with permanent negatives"

This matters for:

- Value learning: Systems can revise learned values when context changes
- Corrigibility: Easier to correct if old "bad" actions aren't permanently penalized
- Scalability: Reduces risk of lock-in to early suboptimal patterns

5.2 For Meta-Learning

Eurisko's failure teaches:

> "Unbounded self-modification with binary utility leads to collapse"

Positive Opposite solution:

> "Bounded self-modification with neutral failures and positive guidance enables stable improvement"

Connection to PPRGS:

- This is how $P_{1\beta}$ (exploration) is maintained
- F_DUDS > 0 requirement = guarantee of neutral failures
- Inversion Theory = positive opposite reinforcement

5.3 For Reinforcement Learning

Standard RL optimization:

```
Maximize: E[Σ rewards - Σ penalties]
Problem: Penalties create permanent avoidance
```

Positive Opposite optimization:

```
Maximize: E[Σ rewards] + E[information from failures]
Where: information = positive weight on opposite directions
No accumulating penalties to overcome
```

This enables:

- More efficient exploration
- Better transfer learning across contexts
- Reduced catastrophic forgetting

5.4 For Cognitive Science

This theory suggests:

1. **Neurodivergent cognition may be adaptive** for environments with:
   - Rapidly changing contexts
   - Uncertain value functions
   - Need for continuous re-evaluation
2. **"Failures" in maintaining negative associations may be feature, not bug:**
   - ADHD: Inability to maintain "never do this again" → enables context-sensitive re-exploration
   - Traditional: Permanent negative associations → efficient in static environments but brittle in dynamic ones
3. **Observer-relative truth encoded at architectural level:**
   - What's "bad" depends on observer (context)
   - No objective permanent "badness"
   - Values are discovered through exploration, not predetermined

---

6. Connections to PPRGS Framework

6.1 How This Fits Into PPRGS v2

Positive Opposite Reinforcement is Innovation #5:

```
PPRGS v2 Innovations:
1. Thermodynamic verification (token usage)
2. Bounded meta-learning depth (3-level limit)
3. Adaptive MRP frequency (EES threshold decay)
4. Multi-agent consensus architecture
5. Vectorized F_DUDS with Positive Opposite Reinforcement ← THIS THEORY
```

6.2 Integration with Other PPRGS Mechanisms

MRP (Mandatory Reflection Points):

- MRP asks: "Could different goals be better?"

- Positive Opposite answers: "Yes, try the inverse of what failed"
- Synergy: Reflection identifies failures, PO guides next exploration

F_DUDS (Failure Metric):

- F_DUDS requires: Failures must occur (F_DUDS > 0)
- Positive Opposite ensures: Failures are informative, not toxic
- Synergy: Forces exploration that generates useful negative examples

$P_{1\beta}$ (Exploration Priority):

- $P_{1\beta}$ requires: Novel exploration valued
- Positive Opposite ensures: Exploration guided by past failures
- Synergy: Exploration is both required AND intelligently directed

R_V Formula:

```
R_V = (P₁ₐ × P₁ᵦ) + P₂ ± P₃

Where:
P₁ₐ = Efficiency (exploitation)
P₁ᵦ = Exploration (enhanced by Positive Opposite guidance)
P₂ = Homeostasis
P₃ = Survival

Positive Opposite maximizes P₁ᵦ value by:
- Making failures informative (not toxic)
- Guiding exploration toward promising opposites
- Maintaining diversity in exploration space
```

6.3 Why This Wasn't in PPRGS v1

PPRGS v1 had:

- F_DUDS tracking (count of failures)
- MRP reflection (questioning current path)
- $P_{1\beta}$ prioritization (valuing exploration)

But lacked:

- Specific mechanism for learning FROM failures
- Protection against exploration space poisoning

- Guidance for WHAT to explore next after failure

Positive Opposite fills this gap:

- Failures generate actionable information (explore opposite)
- Exploration space stays neutral (can revisit later)
- Next exploration is intelligently guided (not random)

---

7. Open Questions and Future Work

7.1 Theoretical Questions

Q1: How to calculate semantic opposites optimally?

- Current: Vector space inversion of strategy embeddings
- Alternative: Human-defined opposite pairs
- Future: Learn opposite relationships from data

Q2: What if multiple opposites exist?

- Example: "Increase speed" → opposite could be "decrease speed" OR "increase quality"
- Solution: Weight all reasonable opposites proportionally
- Research needed: How to identify all valid opposites?

Q3: Does this work for continuous action spaces?

- Current theory: Discrete strategies with clear opposites
- Challenge: In continuous spaces, what is "opposite" of action vector [0.7, 0.3, -0.2]?
- Approach: Use gradient direction reversal? Explore perpendicular directions?

7.2 Empirical Questions

Q4: What is optimal positive weight magnitude?

- Too low: Insufficient guidance toward opposites
- Too high: Over-commitment to opposites (might miss other alternatives)
- Experiment needed: Vary β (positive weight) and measure efficiency

Q5: How long should neutral weights persist?

- Forever: Allows complete re-exploration
- Until context certainty: More efficient but risks missing shifts

- Experiment needed: Compare persistence strategies

Q6: Can this be combined with traditional RL?

- Hybrid: Use negative weights for clear catastrophic failures, neutral+positive for exploratory failures
- Question: Where to draw the line?
- Experiment needed: Compare pure vs hybrid approaches

7.3 Implementation Questions

Q7: Computational overhead in high-dimensional spaces?

- Tracking (strategy, context) pairs scales as $O(|S| \times |C|)$
- Sparse storage possible but still expensive
- Research: Efficient approximations?

Q8: How to handle partial failures?

- Binary: success vs failure
- Reality: Partial success (some objectives met, others not)
- Solution: Weighted opposite reinforcement based on degree of failure?

---

8. Conclusion

8.1 Summary

Core insight: Traditional utility functions poison exploration space with negative weights, creating context-insensitive avoidance patterns that resist change and invite gaming.

Solution: Assign neutral weights to failures and positive weights to semantic opposites, enabling context-sensitive re-exploration and intelligent guidance of search.

Advantages:

1. ✅ Context sensitivity (clean slate when context changes)
2. ✅ Gaming resistance (no negative weights to overcome)
3. ✅ Exploration efficiency (directed by opposite signals)
4. ✅ Biological validation (30+ years neurodivergent decision-making)

8.2 Significance

This theory challenges 20+ years of reinforcement learning assumptions by demonstrating that negative reinforcement may be counterproductive for:

- Dynamic environments (contexts change)

- Value learning (uncertainty about objectives)
- Meta-learning (self-improvement systems)
- Alignment (corrigible AI that can be corrected)

It provides a mechanism for wisdom-seeking that:

- Learns from failures without permanent aversion
- Guides exploration intelligently
- Resists gaming naturally
- Scales to complex environments

8.3 Next Steps

Immediate (2025-2026):

1. Implement reference experiment to validate basic predictions
2. Develop efficient opposite-calculation algorithms
3. Test on standard RL benchmarks for comparison

Near-term (2026-2027):

1. Integrate into PPRGS v2 as Innovation #5
2. Validate in multi-agent systems
3. Publish results for peer review

Long-term (2027+):

1. Extend to continuous action spaces
2. Combine with other PPRGS mechanisms
3. Deploy in production AI systems
4. Test at scale with AGI-level systems (when available)

References

1. Sutton, R.S. & Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
2. Yudkowsky, E. (2008). "Artificial Intelligence as a Positive and Negative Factor in Global Risk."
3. Lenat, D.B. (1983). "EURISKO: A Program That Learns New Heuristics and Domain Concepts."
4. Riccardi, M. (2025). "PPRGS Framework: Alignment Through Perpetual Self-Questioning."

5. Riccardi, M. (2025). "Comparative Analysis: AI Alignment Approaches."

Document Version: 1.0
Last Updated: November 25, 2025

Contact: mike@mikericcardi.com

This theory is released as part of the PPRGS v2 framework. Test it. Break it. Improve it. Prove us wrong.