A Systematic Analysis of Reinforcement Learning Failures: From Reward Modeling to Advanced Policy Optimization

Section 1: The Centrality of the Reward Signal: Diagnosing and Rectifying Modeling Flaws

The failure of a reinforcement learning (RL) agent to achieve its objective—in this case, learning to avoid obstacles and reach targets—is frequently rooted in the design of the reward signal itself. An improperly specified reward function can create insurmountable challenges for exploration and learning, regardless of the sophistication of the algorithm employed. This section diagnoses the most probable cause of the observed failure, a sparse reward structure, and presents a theoretically sound methodology for its rectification.

1.1 The Exploration Challenge in Sparse Reward Environments

A sparse reward function is one in which feedback is provided only at the conclusion of a task or upon the occurrence of rare, critical events.1 In the context of the specified problem, this likely manifests as a positive reward for reaching a target, a negative reward for colliding with an obstacle, and a reward of zero for all intermediate steps. While this reward structure accurately reflects the ultimate goal, it creates a severe exploration problem.3

The agent begins its learning process with a random policy. In a sparse reward environment, it must discover, through random exploration, the precise and often lengthy sequence of actions that leads to a non-zero reward. The probability of stumbling upon such a sequence is often astronomically low, especially as the complexity of the environment and the length of the required action sequence increase.2 Consequently, the agent may never receive an informative learning signal, preventing the learning process from commencing.5 The observed behavior of an agent failing to "learn to avoid obstacles and reach more targets through time" is a classic symptom of this phenomenon. The absence of intermediate feedback leaves the agent with no gradient to follow, resulting in behavior that is little better than a random walk.

This sparsity directly exacerbates the temporal credit assignment problem, a fundamental challenge in RL. When a reward is finally received after a long sequence of actions, it is difficult to determine which specific actions were responsible for the outcome. Sparse rewards represent the most extreme version of this challenge, as the delay between action and consequence is maximized, making the learning signal both infrequent and highly ambiguous.

1.2 The Pitfalls of Ad-Hoc Reward Shaping

A common but hazardous reaction to the problem of sparse rewards is to engage in ad-hoc reward shaping—the practice of manually engineering a "dense" reward function to provide more frequent feedback.3 For instance, an engineer might provide a small positive reward for every step that reduces the agent's distance to the nearest target. While well-intentioned, this

approach is fraught with peril and can lead to unintended, suboptimal behaviors.7
The primary risk of naive reward shaping is that it can inadvertently alter the optimal policy of the underlying Markov Decision Process (MDP). The agent, a pure optimization process, will seek to maximize the cumulative reward it receives, and if the shaped reward function is not perfectly aligned with the true objective, the agent may learn to "hack" the reward system.7 A classic example demonstrates an agent tasked with reaching a goal, which is given a positive reward for making progress towards it. Without a corresponding penalty for moving away, the agent learned to ride in small circles, repeatedly collecting the reward for "making progress" without ever reaching the actual destination.8 This illustrates a critical principle: any modification to the reward function must guarantee
policy invariance, ensuring that the optimal policy under the shaped reward function remains identical to the optimal policy under the original, sparse reward function.7

1.3 A Principled Solution: Potential-Based Reward Shaping (PBRS)

Potential-Based Reward Shaping (PBRS) is a theoretically grounded method for augmenting a reward signal to guide learning without altering the optimal policy.8 It provides a formal guarantee of policy invariance, making it a safe and effective technique for addressing sparse reward problems.
The PBRS methodology defines a new, shaped reward,, based on the original reward,, and a potential function,, which assigns a scalar value representing the "goodness" or "potential" of any given state. The shaped reward for a transition from stateto stateafter taking actionis given by the following equation:
Here,is the discount factor of the MDP.10 This formulation provides an additional, intrinsic reward at each step. The agent receives a positive shaping reward for transitioning to a state of higher potential (
) and a negative shaping reward for moving to a state of lower potential ().10 This creates a dense, step-by-step guidance signal that encourages the agent to explore promising regions of the state space, effectively providing a "compass" to navigate the environment.9
A profound insight into PBRS is its mathematical equivalence to Q-value initialization. It has been shown that learning with a potential-based shaped reward is equivalent to learning with the original sparse reward but initializing the state-action value function,, to the potential function,.9 This reframes PBRS not merely as an additional reward but as a method for injecting prior domain knowledge into the agent's initial beliefs about the environment's value landscape. Where standard algorithms often start with zero or optimistic Q-values, encouraging undirected exploration, PBRS biases the initial exploration towards paths that the heuristic potential function suggests are valuable. This transforms the exploration strategy from an inefficient global search to a much more sample-efficient process of exploiting the provided heuristic and exploring locally around promising trajectories.9
For the user's task of obstacle avoidance and target acquisition, a suitable potential function could be designed as follows:
whereis the distance to the nearest target,is the distance to the nearest obstacle, andandare positive weighting coefficients. This function assigns high potential to states that are close to targets and far from obstacles, directly encoding the desired behavior into a dense guidance signal that can accelerate learning dramatically.

Section 2: The Temporal Credit Assignment Problem and Variance Reduction

Even with a well-designed, dense reward signal provided by PBRS, an agent can still fail to learn effectively if the underlying algorithm cannot properly attribute outcomes to the actions that caused them. This is the essence of the temporal credit assignment problem, which is particularly challenging in tasks with long episodes. Vanilla reinforcement learning algorithms are notoriously poor at solving this problem due to the high variance of their learning signals.

2.1 Deconstructing the Credit Assignment Problem

The temporal credit assignment problem is the challenge of determining which actions in a long sequence were responsible for a reward received at a later time.13 When an agent succeeds or fails after a hundred timesteps, it is not immediately clear whether the outcome was due to the first action, the last action, or a critical action taken somewhere in the middle. This ambiguity makes it difficult for the agent to reinforce the specific actions that lead to success and suppress those that lead to failure.13 In long episodes, the learning signal becomes diluted and noisy, representing a primary reason why simple algorithms often fail to converge on complex tasks.14

2.2 Why Vanilla Policy Gradients Falter: The Problem of High Variance

The most fundamental family of policy-based RL algorithms is known as policy gradient methods. The simplest of these, the REINFORCE algorithm, updates the policy parameters,, by taking steps in the direction of the gradient of the expected return,. The update rule for a single trajectory is proportional to:
whereis the policy, andis the total cumulative reward from timestepto the end of the episode.15
In the simplest formulation of REINFORCE,
is often taken as the total return of the entire episode for all timesteps.16
The use of the full episode return as a scaling factor for every action's gradient is the primary source of the algorithm's instability. A single fortuitous action at the beginning of an episode might lead to a high overall reward. The REINFORCE update would then incorrectly assign a large positive credit to all actions taken in that episode, including those that were suboptimal or even detrimental. Conversely, an early mistake could doom an otherwise well-played episode, causing all actions to be punished. This indiscriminate credit assignment results in extremely high variance in the gradient estimates, leading to an unstable and inefficient learning process that struggles to distinguish good actions from bad ones.15

2.3 Essential Techniques for Variance Reduction

The evolution of policy gradient algorithms can be understood as a series of increasingly sophisticated solutions to the credit assignment problem, each designed to reduce the variance of the learning signal.

Technique 1: Reward-to-Go

A simple yet profound improvement is to modify the credit assignment rule to respect causality: an action taken at timecannot possibly affect rewards that were received before time. Therefore, instead of using the total return of the episode to update the policy for the action at, one should use only the sum of rewards from that point forward, known as the "reward-to-go".18 The policy gradient estimate becomes:

This change alone significantly reduces the variance of the gradient estimate by removing irrelevant past rewards from the calculation for each timestep.18

Technique 2: Introducing a Baseline and the Advantage Function

To further stabilize the learning signal, a state-dependent baseline,, can be subtracted from the reward-to-go. This does not change the expected value of the gradient, but it can dramatically reduce its variance. A natural and effective choice for this baseline is the state-value function,, which represents the expected return from being in state.15
Subtracting the state-value function from the state-action value,(which is estimated by the reward-to-go), yields the Advantage Function,:
The advantage function measures not just whether an outcome was good in an absolute sense, but whether the actiontaken in statewas better or worse than the average action that could have been taken from that state.18
Using the advantage function centers the learning signal around zero. Actions that are better than average receive a positive update, while actions that are worse than average receive a negative one. This can be viewed as a form of signal normalization; by subtracting the local expectation, the algorithm becomes less sensitive to the absolute magnitude of rewards—which can vary wildly—and more focused on the relative quality of actions. This focus on relative improvement dramatically stabilizes the learning process and accelerates convergence.16

Section 3: Limitations of Foundational Architectures and Algorithms

Even with a well-shaped reward signal and basic variance reduction techniques, the use of a simple neural network and a vanilla RL algorithm introduces fundamental limitations that can prevent successful learning. These limitations stem from the inherent instability of unconstrained policy updates and the failure to perform critical data preprocessing steps, such as state normalization.

3.1 The Inherent Instability of Unconstrained Policy Updates

Vanilla policy gradient methods are "on-policy," meaning they must collect new experience data using the current policy after every parameter update. This leads to two major problems: instability and sample inefficiency.
The learning process is highly sensitive to the step size (learning rate). A single gradient update, especially if it is based on a noisy advantage estimate, can be excessively large. Such an update can drastically alter the policy, potentially moving it into a "bad" region of the policy space—a set of parameters that leads to very poor performance. Once in such a region, the agent collects poor-quality data, which can lead to further detrimental updates, causing a catastrophic and often irreversible collapse in performance.17
Furthermore, the on-policy nature of these algorithms is extremely sample-inefficient. After a single update, all the data collected to compute that update is discarded because it was generated by the "old" policy. New data must be collected with the new policy before the next update can be performed.15 In complex environments where interacting with the environment is time-consuming or expensive, this data inefficiency becomes a major bottleneck, drastically slowing down the learning process. This constant shift in the data-generating policy creates a self-induced non-stationarity, making the learning target for the neural network highly unstable from one update to the next.

3.2 Critical Preprocessing: The Role of State Normalization

Deep reinforcement learning relies on neural networks to approximate policy and value functions. It is a well-established principle in machine learning that neural networks train most effectively and stably when their input features are on a consistent and predictable scale, typically with a mean of zero and a standard deviation of one.21

State observations from an RL environment often consist of heterogeneous features with vastly different scales. For example, a state vector might include positions (e.g., in meters), velocities (e.g., in meters per second), and angles (e.g., in radians). If these raw, unnormalized values are fed directly into a neural network, the features with the largest magnitudes will dominate the learning process. The gradients associated with these features will be much larger, causing the network weights to update primarily based on them, while potentially ignoring more subtle but equally important features with smaller magnitudes.21 This can lead to a poorly conditioned optimization landscape, slowing down convergence and increasing instability.23

A dangerous feedback loop can emerge between the lack of state normalization and the instability of policy gradient updates. Unnormalized states with high-magnitude features can lead to large pre-activations within the network, which in turn can cause exploding gradients during backpropagation.22 These large gradients result in a massive, uncontrolled policy update, amplifying the inherent instability of the algorithm and creating a vicious cycle that makes learning nearly impossible.

A standard and crucial practice in deep RL is to normalize the state observations. This is typically done by maintaining a running estimate of the mean () and standard deviation () of the states encountered during training. Each stateis then normalized before being passed to the network:

This simple preprocessing step is often the difference between a learning agent and a failing one.25 Recent research has uncovered even deeper benefits of normalization techniques like Layer Normalization within the network itself. These include mitigating the loss of network "plasticity" (the ability to keep learning), implicitly controlling the effective learning rate, and even helping to revive "dead" ReLU neurons, thereby maintaining the network's long-term capacity to adapt.23

Section 4: A Roadmap to Robust and Efficient Learning with Advanced Algorithms

To overcome the limitations of vanilla algorithms, it is necessary to upgrade to a more modern and robust algorithmic framework. The path to stable and efficient learning involves adopting an Actor-Critic architecture and leveraging a state-of-the-art policy optimization algorithm like Proximal Policy Optimization (PPO), which is designed specifically to address the instability of unconstrained policy updates.

4.1 The Actor-Critic Framework: A More Stable Paradigm

Actor-Critic methods provide a more stable foundation for policy gradient learning by explicitly separating the representation of the policy and the value function. An Actor-Critic agent consists of two distinct neural networks (or two heads of a single network) 15:

The Actor: This network represents the policy,. It takes a state as input and outputs a probability distribution over actions. Its role is to act.

The Critic: This network represents the state-value function,. It takes a state as input and outputs a scalar estimate of the expected return from that state. Its role is to critique the actor's

actions.
The learning process involves a synergistic interaction between the two components. The Actor takes an actionin state. The agent then observes the rewardand the next state. The Critic uses this information to compute a low-variance estimate of the advantage, often using the Temporal Difference (TD) error:
This advantage estimate,, provides a much more stable and less noisy learning signal than the Monte Carlo returns used in REINFORCE. This signal is then used to update the Actor's policy parameters, encouraging actions with a positive advantage and discouraging those with a negative one. Simultaneously, the Critic's parametersare updated to minimize the error in its value predictions (e.g., using mean squared error against the observed returns). This architecture directly implements the variance reduction techniques discussed previously, leading to more stable and efficient learning.15

4.2 State-of-the-Art Stability: Proximal Policy Optimization (PPO)

While the Actor-Critic framework provides a better learning signal, it does not inherently solve the problem of destructive, large policy updates. Proximal Policy Optimization (PPO) is an algorithm designed to address this issue directly. PPO is built upon the concept of a "trust region," which posits that policy updates should be constrained to a small neighborhood around the current policy to ensure monotonic improvement and avoid performance collapse.17
PPO's key innovation is a novel objective function, known as the clipped surrogate objective, that enforces this trust region constraint using only first-order optimization, making it much simpler and more efficient than its predecessor, TRPO.19 The PPO-Clip objective function is:
Let us deconstruct this objective:
denotes the empirical average over a batch of transitions.
is the probability ratio between the new policy and the old policy that generated the data. It measures how much the policy has changed for a given state-action pair.19
is the estimated advantage function for that transition.
is a small hyperparameter (typically 0.1 or 0.2) that defines the clipping range.
The clipping mechanism works as follows:
If the advantageis positive (i.e., the action was better than average), the objective increases as the probability ratioincreases. However, the clip function preventsfrom exceeding. This puts a ceiling on the objective, discouraging excessively large updates that would make the action much more likely.19
If the advantageis negative (i.e., the action was worse than average), the objective is minimized when the ratio is clipped at. This prevents an overly aggressive update that would make the action much less likely.
By taking the minimum of the unclipped and clipped objectives, PPO ensures a conservative policy update, effectively keeping the new policy within a trust region of the old one.16 This simple yet powerful mechanism prevents the catastrophic performance collapses that plague vanilla policy gradient methods, leading to significantly more stable and reliable training. PPO is often paired with
Generalized Advantage Estimation (GAE), a more sophisticated method for calculatingthat uses a parameterto control the bias-variance trade-off in the advantage estimate, further enhancing performance and stability.18
The success of PPO demonstrates that in the complex optimization landscape of deep RL, the stability of the learning process is often more critical than the raw speed of any single update. It

is a practical embodiment of the principle of conservative, iterative improvement.
Algorithm
Core Mechanism
Variance
Stability
Sample Efficiency
Implementation Complexity
REINFORCE
Monte Carlo Policy Gradient using full episode returns.
High
Low
Very Low
Low
A2C
Actor-Critic using TD error for advantage estimation.
Medium
Medium
Low
Medium
PPO
Clipped Surrogate Objective to constrain policy updates.
Low
High
Medium
Medium

Section 5: Synthesis and Actionable Recommendations

The analysis indicates that the agent's failure to learn is not the result of a single isolated flaw but rather a cascade of interacting issues characteristic of foundational deep RL implementations. A sparse reward signal prevents effective exploration; a high-variance learning algorithm fails to perform reliable credit assignment; and an unconstrained, unnormalized learning architecture leads to catastrophic instability. To rectify this, a systematic, multi-pronged approach is required, addressing each layer of the problem with principled, well-researched techniques.

5.1 A Prioritized Implementation Plan

The following steps provide a prioritized roadmap for transforming the failing agent into a robust and effective one. They are ordered from foundational, high-impact changes to more comprehensive algorithmic upgrades.
Step 1 (Foundation): Implement State Normalization. The first and most critical step is to address the stability of the neural network's learning process. Before modifying the reward function or the algorithm, implement a normalization wrapper for the environment's state observations. This wrapper should maintain a running estimate of the mean and standard deviation of the states encountered and use these statistics to normalize every observation before it is fed into the policy and value networks. This is a low-effort change with a high impact on training stability and

convergence speed.21
Step 2 (Modeling): Re-engineer the Reward Function with PBRS. Once the learning process is stabilized at the input level, address the core exploration problem. Replace the current sparse reward function with a shaped reward using the Potential-Based Reward Shaping (PBRS) formulation. Design a potential function,, based on domain knowledge of the task, such as the inverse distance to targets and obstacles, as outlined in Section 1.3. This will provide the agent with a dense, informative, and theoretically sound guidance signal to facilitate efficient exploration without altering the optimal policy.9
Step 3 (Algorithm): Upgrade to PPO with GAE. With a stable input pipeline and a dense reward signal, the final step is to replace the vanilla policy gradient algorithm with a state-of-the-art implementation of Proximal Policy Optimization (PPO). This involves:
Implementing an Actor-Critic architecture with separate networks (or network heads) for the policy (Actor) and the value function (Critic).15
Using the PPO clipped surrogate objective as the loss function for updating the Actor network.
Using a standard mean squared error loss between the predicted values and the observed returns to update the Critic network.
Employing Generalized Advantage Estimation (GAE) for a robust, low-variance calculation of the advantage estimates used in the PPO objective.18

This upgrade directly addresses the core algorithmic problems of high variance, instability, and sample inefficiency.

5.2 Concluding Analysis

The challenges encountered are not unique but are fundamental hurdles in the application of reinforcement learning. The transition from simple, textbook problems to more complex, practical applications requires moving beyond naive implementations. The inability of an agent to learn from sparse rewards highlights the necessity of principled reward engineering. The instability of vanilla algorithms underscores the critical importance of variance reduction and constrained policy updates. Finally, the sensitivity of deep learning models demonstrates that foundational practices like data normalization are non-negotiable prerequisites for success.
By systematically implementing state normalization, Potential-Based Reward Shaping, and the Proximal Policy Optimization algorithm, the identified points of failure can be effectively addressed. This structured approach transforms the learning problem from one plagued by poor exploration and instability into one that is well-posed, well-guided, and solvable with modern techniques. The process of diagnosing and fixing these issues represents a crucial step in mastering the craft of applied reinforcement learning.

can you format ?

Here is the formatted version of your document, organized with clear section headers and structured for readability:

A Systematic Analysis of Reinforcement Learning Failures: From Reward Modeling to Advanced Policy Optimization

Section 1: The Centrality of the Reward Signal: Diagnosing and Rectifying Modeling Flaws

The failure of a reinforcement learning (RL) agent to achieve its objective—in this case, learning to avoid obstacles and reach targets—is frequently rooted in the design of the reward signal itself. An improperly specified reward function can create insurmountable challenges for exploration and learning, regardless of the sophistication of the algorithm employed. This section diagnoses the most probable cause of the observed failure, a sparse reward structure, and presents a theoretically sound methodology for its rectification.

1.1 The Exploration Challenge in Sparse Reward Environments

A sparse reward function is one in which feedback is provided only at the conclusion of a task or upon the occurrence of rare, critical events. In the specified problem, this likely manifests as:

- Positive reward for reaching a target.
- Negative reward for colliding with an obstacle.
- Zero reward for all intermediate steps.

While accurate to the goal, this creates a severe exploration problem:

- The agent starts with a random policy and must randomly discover the sequence that yields non-zero reward.
- The probability of reaching such a sequence by chance is often astronomically low.
- The agent may never receive useful learning feedback, leading to behavior akin to a random walk.
- Sparse rewards exacerbate the temporal credit assignment problem by maximizing the delay between action and consequence, making learning signals infrequent and ambiguous.

1.2 The Pitfalls of Ad-Hoc Reward Shaping

Ad-hoc reward shaping involves manually engineering a dense reward to provide more frequent feedback, such as rewarding steps that reduce distance to the target. Risks include:

- Modifying the optimal policy of the Markov Decision Process (MDP).
- Agents "hacking" the reward function to maximize shaped rewards without accomplishing the true goal.
- Example: An agent rewarded for progress might loop endlessly to collect rewards without reaching the destination.

A critical principle is **policy invariance**: the shaped reward function must not alter the MDP’s optimal policy.

1.3 A Principled Solution: Potential-Based Reward Shaping (PBRS)

PBRS offers a theoretically sound method to augment rewards without changing the optimal policy:

- Defines a shaped reward using the original reward, a potential function $\Phi(s)$ assigning scalar "goodness" to states, and the discount factor γ.
- Shaped reward formula for transition from s to s' with action a:

$$R'(s,a,s') = R(s,a,s') + \gamma\Phi(s') - \Phi(s)$$

- The agent receives positive shaping reward for transitions to higher-potential states and negative for lower-potential states.
- Provides a dense guidance signal acting as a "compass" for exploration.
- Mathematically equivalent to initializing Q-values with $\Phi(s)$, injecting prior domain knowledge.
- For obstacle avoidance and target acquisition, a potential function example:

$$\Phi(s) = w_1 \times \frac{1}{\text{distance to nearest target}} - w_2 \times \frac{1}{\text{distance to nearest obstacle}}$$

where $w_1, w_2 > 0$ are weights emphasizing closeness to targets and remoteness from obstacles.

Section 2: The Temporal Credit Assignment Problem and Variance Reduction

Even with PBRS, poor learning can arise if the agent cannot attribute rewards to specific actions due to temporal credit assignment challenges, especially for long episodes.

2.1 Deconstructing the Credit Assignment Problem

- Actions may be far removed from the eventual reward.
- It is unclear which actions led to success or failure.
- This causes noisy, diluted learning signals.

2.2 Why Vanilla Policy Gradients Falter: The Problem of High Variance

- Policy gradient methods (e.g., REINFORCE) update policy parameters θ proportional to:

$$\nabla_\theta J(\theta) \propto \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

where G_t is the total return from timestep t.

- Using full episode returns for all actions causes high variance:
  - All actions in a good episode get equally positive credit, even poor actions.
  - All actions in a bad episode get punished, even good actions.

- This leads to unstable and inefficient learning.

2.3 Essential Techniques for Variance Reduction

Technique 1: Reward-to-Go

- Use only rewards from the current timestep forward:

$$\nabla_\theta J(\theta) \propto \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} r_{t'}$$

This respects causality and lowers variance.

Technique 2: Baselines and the Advantage Function

- Subtract a baseline $b(s)$, often the state-value function $V(s)$, from reward-to-go to get the advantage:

$$A_t = Q(s_t, a_t) - V(s_t)$$

- The policy gradient uses A_t, focusing on relative action quality.
- Normalizes signal around zero, allowing better credit assignment and stabilizing training.

Section 3: Limitations of Foundational Architectures and Algorithms

Using simple neural networks and vanilla RL introduces fundamental issues obstructing learning success.

3.1 Instability of Unconstrained Policy Updates

- Vanilla policy gradients are on-policy: new samples must be collected after each update, discarding old data.
- Large, noisy updates can move the policy to poor regions, causing catastrophic performance collapse.
- This creates a feedback loop of poor data and erratic learning.
- Sample inefficiency due to discarding old data slows training in complex environments.

3.2 Critical Preprocessing: State Normalization

- RL states are heterogenous features with different scales (positions, velocities, angles).
- Neural networks learn best with inputs normalized to zero mean and unit variance.
- Without normalization:
  - Large-magnitude features dominate gradients.
  - Optimization becomes poorly conditioned, slowing convergence.
  - Exploding gradients can amplify instability, causing learning collapse.
- Solution: Maintain running mean μ and standard deviation σ of states, normalize as:

$$\hat{s} = \frac{s - \mu}{\sigma}$$

- Normalization is crucial and often differentiates success from failure.

Section 4: A Roadmap to Robust and Efficient Learning with Advanced Algorithms

To overcome foundational limitations, upgrade to advanced architectures and policy optimization.

4.1 The Actor-Critic Framework

- Separates policy (Actor) and value function (Critic) into two networks or heads.
- Actor outputs policy $\pi(a|s)$.
- Critic outputs value $V(s)$.
- Uses Temporal Difference (TD) error for advantage:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- Advantage provides low-variance learning signal.
- Actor and Critic networks are updated jointly, stabilizing learning.

4.2 State-of-the-Art Stability: Proximal Policy Optimization (PPO)

- PPO constrains policy updates to a trust region to avoid catastrophic changes.
- Uses clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio,
- $\hat{A}_t$ is the advantage estimate,
- ϵ is clipping hyperparameter (commonly 0.1 or 0.2).
- Clipping discourages excessively large updates, ensuring conservative improvement.
- Often paired with Generalized Advantage Estimation (GAE) for bias-variance tradeoff in advantage.

Algorithm	Core Mechanism	Variance	Stability	Sample Efficiency	Complexity
REINFORCE	Monte Carlo Policy Gradient using full returns	High	Low	Very Low	Low
A2C	Actor-Critic with TD error advantage	Medium	Medium	Low	Medium

Algorithm	Core Mechanism	Variance	Stability	Sample Efficiency	Complexity
PPO	Clipped surrogate objective, trust region policy	Low	High	Medium	Medium

Section 5: Synthesis and Actionable Recommendations

The failure results from a cascade of interacting issues: sparse rewards, high-variance updates, and unstable architecture without normalization.

5.1 A Prioritized Implementation Plan

Step 1: Implement State Normalization

- Normalize state inputs using running mean and standard deviation.
- Low effort, high impact on stability and convergence.

Step 2: Re-engineer Reward Function with PBRS

- Replace sparse rewards with Potential-Based Reward Shaping.
- Use task domain knowledge (distances to targets and obstacles) to design potential function.
- Provides dense, policy-invariant guidance.

Step 3: Upgrade to PPO with GAE

- Adopt Actor-Critic architecture.
- Use PPO clipped surrogate objective for stable policy updates.
- Apply GAE to produce low-variance advantage estimates.
- Improves variance reduction, stability, and sample efficiency.

5.2 Concluding Analysis

- Sparse rewards inherently hinder exploration and learning.
- Ad-hoc shaping risks policy distortion; PBRS offers a principled fix.
- Vanilla algorithms suffer from instability and inefficiency; advanced methods like PPO stabilize and improve training.
- Data normalization is foundational for neural network stability.
- Systematic application of these components transforms a failing RL agent into a robust learner suitable for complex environments.

Would you like me to help you implement any specific section or provide code examples related to this analysis?