Nick Clark · Published April 25, 2026 · PDF

How to Make an AI Agent Forecast Its Own Likely Failure Modes

You want an agent that, before it commits, can tell you that this action is likely to fail, that no viable path forward exists, or that it is about to act on something it only imagined. This guide describes an architecture for exactly that, disclosed in United States Patent Application 19/647,395. It is a design you implement yourself, not a shipping library. It centers on the Forecasting Engine inventive step: a self-forecasting cycle in which the agent projects a multi-dimensional trajectory of its own future state, evaluates candidate actions and its own likely failure modes speculatively before commitment, and can conclude that the safest move is not to act.

What You Are Building

You are building an agent that, at each decision point, projects the futures available to it, evaluates each one, and can arrive at a specific and useful conclusion: not just "here is my best move" but "every move I can see leads somewhere bad," or "I have no move that satisfies the goal without violating my constraints." In other words, an agent that forecasts its own likely failure modes before it acts on them, and treats that forecast as a first-class input to whether it acts at all.

The developer who searches for this usually has an agent that already plans and calls tools, and has watched it confidently march into a task it was never going to complete, or take an irreversible action whose downside only became visible afterward. What you want is for the agent to surface the failure ahead of time: to notice that the space of viable plans is empty, or that a tempting branch would break a hard constraint, and to change its behavior because of what it foresaw.

The architecture here comes from the Forecasting Engine disclosed in United States Patent Application 19/647,395. Its core idea is a self-forecasting cycle: the agent projects hypothetical trajectories of its own future state, classifies each one, and derives a viability signal from the classification. This is an architecture you build yourself, not a package you install.

Why the Obvious Approaches Fall Short

The usual ways of getting an agent to anticipate failure each help a little and leave the same gap.

Ask the model to list its risks. You can prompt a model to enumerate what might go wrong before it acts. This produces plausible-sounding text, but the text has no enforced relationship to what the agent does next. Nothing structurally prevents the agent from acting against its own stated risk list, and the enumeration is not tied to the actual state transitions the agent would perform.

Post-hoc error handling and retries. Try/except, timeouts, and retry loops catch failures after they occur. That is necessary engineering, but by definition it acts once the failure has already happened. It gives you recovery, not foresight, and it cannot stop an irreversible action before commitment.

Statistical rollouts with a value estimate. Methods such as Monte Carlo Tree Search sample many futures and estimate their value statistically over random rollouts. That is a legitimate planning technique. The disclosed approach differs in kind: the specification describes simulation that operates deterministically on defined structural fields, produces reproducible projected outcomes, and is constrained by continuity and policy compatibility at every step rather than evaluated statistically over random samples. A probability estimate tells you a branch is unlikely to pay off; it does not give you a reproducible, inspectable reason a branch is foreclosed.

The gap common to all three is that speculation and commitment are not architecturally separated, and there is no defined signal that means "my speculative reasoning has found no way through." Without that signal, an agent that cannot succeed keeps acting anyway.

The Architecture

The disclosed design turns "forecast your own failure" into three concrete structures: a separated place to imagine futures, a fixed way to classify each imagined future, and a signal derived from those classifications.

A planning graph the agent can safely fail inside. The forecasting engine constructs a planning graph whose root node is the agent's current verified state and whose branches each represent a distinct hypothetical trajectory: a sequence of speculative mutations, projected environmental responses, and projected secondary effects on the agent's own affective and integrity fields. The specification is explicit that the simulation in Phase 2 of the cycle applies these speculative mutations to a sandboxed copy of the agent's state, not to verified execution memory. This is what makes failure forecasting safe: the agent can drive a branch all the way to a bad outcome and learn from it without that bad outcome ever touching committed state.

A six-phase cycle that scores each candidate future. The specification describes a forecasting execution cycle invoked at each cognitive decision point, comprising six sequential phases: initialization (read current verified state, build or refresh the planning graph); speculative mutation simulation (project each branch deterministically in the sandbox); slope projection and validation (check whether the branch maintains trust slope continuity or would produce a discontinuity); policy compatibility check (whether the branch's mutations are admissible under the agent's governance constraints); emotional reinforcement tagging (whether the projected outcome aligns with or is aversive to the agent's affective state); and branch marking. Each phase is a distinct dimension along which a candidate future can turn out badly, and each records why.

A four-way classification that names the failure. After evaluation, every branch receives one of four labels from the disclosed taxonomy. An *eligible* branch passed slope validation, satisfied policy, and was positively or neutrally reinforced; it is a viable candidate. An *introspective* branch is slope-eligible and policy-compatible but negatively reinforced: it is structurally viable but aversive, and it is retained precisely so the agent can reason about why a future is undesirable. A *delegable* branch is viable for a different agent but better handed off. A *pruned* branch failed slope validation, failed policy, or exceeded resource thresholds. This taxonomy is the agent's vocabulary for its own failure modes: a policy-incompatible pruned branch is a foreseen constraint violation; a slope-ineligible branch is a foreseen provenance break; an all-introspective graph is a foreseen aversive outcome with no acceptable alternative.

The negative viability signal. The specification describes the key behavior for self-forecast of failure directly. When the forecasting engine evaluates the active planning graph and finds that all branches have been classified as pruned, introspective, or slope-ineligible, so that no eligible branch exists, it transmits a negative viability signal to the confidence governor. That signal states that the agent's speculative reasoning has exhausted the space of hypothetical futures and found no path from the current state to one that satisfies the agent's intent through slope-eligible, policy-compatible execution.

The confidence governor reduces the agent's confidence metric in response, and the reduction moves the agent out of execution. This is the mechanism by which foreseeing failure changes behavior instead of merely reporting it.

What the agent does with a foreseen failure. The specification describes the resulting non-executing cognitive mode as comprising continued planning graph construction with modified parameters (broader search, longer temporal horizon, relaxed affective biases), inquiry generation (formulating questions to human operators, external sources, or other agents that might supply the missing information), and delegation exploration (evaluating whether a branch that is non-viable for this agent is viable for another). The forecast of failure becomes a request for what would make success possible.

A distinct failure mode: acting on what was only imagined. The specification also names a pathological state worth designing against explicitly. The containment layer keeps speculative planning-graph content structurally separate from verified execution memory; its failure is described as the delusion boundary condition, in which the agent can no longer distinguish what it projected from what actually occurred. The spec lists behavioral coherence monitors that watch for exactly this, for example the agent referencing projected outcomes that have not occurred, or acting on conditions that exist only in a planning graph branch. Forecasting your own failure modes, in this architecture, includes forecasting and detecting the failure of the forecast itself.

How to Approach the Build

The following order lets each layer be tested before the next depends on it. The sketches are illustrative and faithful to the disclosed structure; they are not a library.

1. **Separate speculation from commitment first.** Before any forecasting logic, establish that branches are simulated against a sandboxed copy of state, never against verified memory, and that the only path from speculative to committed runs

through one governed promotion interface. Everything downstream depends on the agent being able to fail inside a branch harmlessly.

2. **Define the branch and its classification labels.** Model a branch as a projected mutation sequence plus the per-dimension results the cycle will fill in. Keep the four labels exactly as disclosed, so failure has a stable vocabulary.

```
Branch:
  mutations:         [ ... ]          # projected, sandboxed
  slope_eligible:    true | false     # Phase 3
  policy_ok:         true | false     # Phase 4
  reinforcement:     positive | neutral | negative   # Phase 5
  label:             eligible | introspective | delegable | pruned
```

3. **Implement the six phases as an ordered, deterministic pipeline.** Initialize from a snapshot of verified state; simulate each branch in the sandbox; run slope validation, then policy compatibility, then reinforcement tagging; then assign a label. Determinism matters: the same input state and mutation sequence should yield the same projected outcome, so a foreseen failure is reproducible and inspectable rather than a one-off sample.
4. **Derive the viability signal from the classification, not from a heuristic.** Compute it structurally: if no branch carries the eligible label, emit a negative viability signal. Resist the temptation to guess viability from a score threshold; the disclosed signal is defined by the absence of any eligible branch.

```
if no branch.label == "eligible":
    emit negative_viability_signal   # -> confidence governor
```

5. **Wire the signal to a real behavior change.** Route the negative viability signal into whatever governs whether your agent acts, so that a foreseen dead end reduces confidence and moves the agent into the non-executing mode. If the signal changes nothing, you have a report, not a forecast.
6. **Implement the non-executing mode as productive, not idle.** On a negative viability signal, do the three disclosed things: broaden the search (wider breadth, longer horizon, relaxed affective bias) and re-run the cycle; generate inquiries that name what information would unlock an eligible branch; and test delegation. This is how the agent turns "I will fail" into "here is what I need."
7. **Retain introspective and pruned branches with their reasons.** Keep negatively-reinforced-but-viable branches for self-examination and keep pruned branches briefly with their rejection annotation. This is what lets the agent explain which failure it foresaw and why, rather than merely refusing.
8. **Add containment monitors last.** Once forecasting works, guard the forecast itself: verify speculative markers, watch boundary crossings, and run behavioral coherence checks for the agent referencing projected-but-unrealized outcomes. This closes the loop by forecasting the failure of forecasting.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and nothing here "just works" out of the box; you implement each layer against your own agent, state model, and constraint system. The method is disclosed in a patent filing; it is not presented as a benchmarked or productized system, and this guide reports no performance numbers because the specification states none.

The design tells you a branch is foreclosed for a defined reason (slope-ineligible, policy-incompatible, or aversive with no eligible alternative). It does not by itself guarantee that your simulation faithfully models the real world; a forecast is only as good as the projected environmental responses you feed it, and a branch that looks eligible in a

poor simulation can still fail in reality. The determinism the spec describes is with respect to defined structural fields and a state snapshot, not a promise that reality is deterministic.

It also presumes the separation the architecture is built on. If you cannot actually keep speculative content out of verified state, or cannot route the viability signal into real behavior, you get the shape of the design without its guarantees. And the approach is aimed at agents that maintain persistent, structured state and reason before acting; it is a poor fit for a stateless, single-shot call where there is no future trajectory to project.

Disclosure Scope

The approach described here is disclosed in United States Patent Application 19/647,395. This guide is educational: it explains an architecture a developer can build, drawing on that filing. It is not a warranty, a specification of a shipping product, or an offer of software, and nothing here should be read as a guarantee of performance or fitness for any purpose. Any third-party technologies mentioned for context are described only to situate the approach, and the disclosed self-forecasting design remains the sole ground truth for what is claimed here.

Forecasting Engine (/forecasting-engine) All 40 steps → (/inventive-steps)

Plan before you act. Contain speculation. Promote only what passes.

Chapter 4 (/patents/19-647395/chapters/forecasting)

PRIMARY TECHNICAL DISCLOSURE

– Forecasting and Executive Graphs in Autonomous Cognitive Systems (/articles/forecasting-and-executive-graphs-in-autonomous-cognitive-systems)

SECONDARY TECHNICAL

- Planning Graphs as First-Class Cognitive Structures (/articles/forecasting-engine/planning-graphs)
- Containment Layer and Delusion Boundary (/articles/forecasting-engine/containment-boundary)
- Branch Classification System (/articles/forecasting-engine/branch-classification)
- Personality Field as Structural Modifier (/articles/forecasting-engine/personality-modifier)
- Executive Engine Multi-Agent Graph Aggregation (/articles/forecasting-engine/executive-aggregation)
- Branch Dormancy and Deferred Promotion (/articles/forecasting-engine/branch-dormancy)
- Proactive Speculative Maintenance (Dream State) (/articles/forecasting-engine/dream-state)
- Planning Graph Archival for Cognitive Forensics (/articles/forecasting-engine/cognitive-forensics)
- Cross-Agent Planning Graph Visibility (/articles/forecasting-engine/cross-agent-visibility)
- Slope-Constrained Speculative Simulation (/articles/forecasting-engine/slope-constrained)
- Structural Separation From Verified Memory (/articles/forecasting-engine/memory-separation)
- Forecasting Engine Architecture (/articles/forecasting-engine/architecture)
- Forecasting Execution Cycle (/articles/forecasting-engine/execution-cycle)
- Emotional Modulation of Planning (/articles/forecasting-engine/emotional-modulation)
- Executive Graph Conflict Resolution (/articles/forecasting-engine/conflict-resolution)
- Planning Graph Delegation and Forking (/articles/forecasting-engine/delegation-forking)
- Temporal Anchoring and Lifecycle Management (/articles/forecasting-engine/temporal-anchoring)
- Forecasting as Coordination Primitive (/articles/forecasting-engine/coordination-primitive)
- Forecasting-Modulated Discovery Traversal (/articles/forecasting-engine/discovery-shaping)
- Forecasting as Confidence Input (/articles/forecasting-engine/confidence-input)
- Integrity-Constrained Forecasting (/articles/forecasting-engine/integrity-constrained)
- Forecasting for Training Curriculum (/articles/forecasting-engine/training-curriculum)
- Biological Signal to Forecasting Coupling (/articles/forecasting-engine/biological-forecasting)
- Substrate-Agnostic Forecasting Deployment (/articles/forecasting-engine/substrate-deployment)
- Uncertainty-Driven Solicitation in the Forecasting Engine (/articles/forecasting-engine/uncertainty-driven-solicitation)
- Cascade Forecasting in the Planning Graph (/articles/forecasting-engine/cascade-forecasting)
- Fleet Behavior Extrapolation (/articles/forecasting-engine/fleet-behavior-extrapolation)

APPLICATIONS · GENERAL

- Cybersecurity Threat Forecasting: Simulating Adversary Trajectories and Predictive Network Reconfiguration as Non-Executing Speculation (/articles/forecasting-engine/cybersecurity-threat-forecasting)
- Surgical Robot Planning AI: Safe Speculative Planning That Never Reaches the Patient (/articles/forecasting-engine/surgical-planning)
- AI Tactical Planning That Explores Adversary Options Without Committing Forces (/articles/forecasting-engine/defense-tactical-planning)
- AI Logistics Planning That Keeps Contingencies Ready: Governed Planning Graphs for Supply Chain Operations (/articles/forecasting-engine/logistics-planning)
- AI Disaster Response Planning: Multi-Scenario Resource Allocation Under Uncertainty (/articles/forecasting-engine/disaster-response-planning)
- Forecasting Engine for Financial Portfolio Planning (/articles/forecasting-engine/financial-portfolio-planning)
- AI Schedule Contingency Management for Construction Project Delay Recovery (/articles/forecasting-engine/construction-project-planning)
- Epidemic Response Planning AI: Multi-Scenario Outbreak Forecasting With an Auditable Decision Record (/articles/forecasting-engine/epidemic-response-planning)
- AI Space Mission Planning: Trajectory Branching and Abort Forecasting Under Light-Time Delay (/articles/forecasting-engine/space-mission-planning)
- Fleet-Scale Active Perception for Autonomous Vehicle Compliance (/articles/forecasting-engine/active-perception-fleet)
- Smart-Grid Load Forecasting With Contained Speculative Planning Graphs (/articles/forecasting-engine/smart-grid-forecasting)

APPLICATIONS · SPECIFIC

- Intuitive Surgical da Vinci vs Governed Forecasting: Trajectories, Not Consequences (/articles/forecasting-engine/intuitive-surgical)
- Anduril Lattice vs Governed Mission Planning: Speculative Containment (/articles/forecasting-engine/anduril)
- Boston Dynamics vs Governed Mission Planning: Motion Is Not Cognition (/articles/forecasting-engine/boston-dynamics)
- Shield AI Hivemind vs Governed Speculative Planning: The Forecasting Engine Axis (/articles/forecasting-engine/shield-ai)
- MuJoCo vs Governed Robot Planning: Contained Speculation Above the Physics Simulator (/articles/forecasting-engine/mujoco)

- NVIDIA Isaac Sim vs Governed Agent Planning: The Forecasting Engine Gap (/articles/forecasting-engine/nvidia-isaac)
- Unity ML-Agents vs Governed Agent Planning at Runtime (/articles/forecasting-engine/unity-ml)
- Gazebo Alternative for Governed Robot Planning: Simulate the World, Contain the Cognition (/articles/forecasting-engine/gazebo)
- Drake vs Governed Robot Planning: Beyond Trajectory Optimization (/articles/forecasting-engine/drake)
- robosuite alternative for governed manipulation planning (/articles/forecasting-engine/robosuite)
- Mobileye REM vs Governed Speculative Planning: Where a Contained Forecasting Layer Sits Above the Roadbook (/articles/forecasting-engine/mobileye-rem)
- Tomorrow.io vs Governed Agent Forecasting: Two Meanings of Forecast (/articles/forecasting-engine/tomorrow-io)
- Skydio vs. a self-forecasting AI agent: trajectory forecasting in flight versus in cognition (/articles/forecasting-engine/skydio)

Forecasting Engine overview → (/forecasting-engine)