Nick Clark · Published July 2, 2026 · PDF

How to Make an AI Agent Simulate the Consequences of an Action Before Acting

You want your agent to look before it leaps: project what a candidate action would do, evaluate it, and only then decide whether to commit. 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, and it centers on the Forecasting Engine inventive step: a forecasting execution cycle in which the agent projects hypothetical trajectories of its own future state, evaluates candidate actions speculatively before executing, and records the forecasting process in lineage before anything is committed.

What You Are Building

You are building an agent that, at each decision point, constructs one or more hypothetical futures for the candidate actions it is considering, simulates each one, evaluates it against several dimensions, and only then decides whether to commit, delegate, defer, or pause. The searcher who lands here usually has an agent that already calls tools and mutates state, and has been burned by it acting first and discovering the consequences later. You want the agent to reason about "what happens if I take this action" as a structured, inspectable step that lives strictly before execution.

The architecture described here comes from the Forecasting Engine disclosed in United States Patent Application 19/647,395. The core idea is a forecasting execution cycle: the agent projects hypothetical trajectories of its own future state, evaluates candidate actions speculatively before executing, and records the forecasting process in its lineage, all before any commitment. This is an architecture to build, not a package to install.

Why the Obvious Approaches Fall Short

The common approaches work, up to a point, but each leaves a structural gap.

Prompt the model to "think first." Chain-of-thought and "reflect before you answer" prompting produce reasoning text, but that text has no architectural relationship to what the agent actually does next. Nothing prevents the agent from acting against its own stated reasoning, and the reasoning is not separated from committed state in any enforced way.

Dry-run flags and sandboxes. Running a tool in a simulated or read-only mode tells you the mechanical result of one call. It does not give you a structured, multi-branch evaluation of alternatives, and the "did the dry run change my real state" boundary is usually a software convention rather than an enforced invariant.

Statistical tree search. Methods such as Monte Carlo Tree Search explore many rollouts and evaluate them statistically over random samples. That is a legitimate technique for many planning problems. The disclosed approach differs in kind: it is described as operating deterministically on defined structural fields, producing reproducible projected outcomes, and constraining each simulation step by continuity and policy compatibility rather than by statistical averaging over random rollouts. If you need auditability and a hard pre-execution filter rather than a probability estimate, random rollouts do not give you that.

The gap common to all three is the same: speculation and commitment are not architecturally separated, so there is no enforced guarantee that "what I imagined" cannot silently become "what I did."

The Architecture

The disclosed design rests on one architectural invariant and one repeating cycle.

The planning graph as a first-class, separated structure. A planning graph is a mutable, directed structure whose root node is the agent's current verified state and whose branches each represent a distinct hypothetical trajectory: a sequence of speculative mutations, delegation outcomes, or environmental transitions the agent is evaluating as a possible future. Critically, the specification describes this graph as maintained in structural separation from the agent's verified execution memory. That separation is described as an architectural invariant, not a namespace or access-control convention: no path exists by which a branch modifies verified memory except through a single governance-controlled promotion interface. The separation is bidirectional. The forecasting engine reads current verified state as the root node once (a snapshot), so evaluations are deterministic with respect to that snapshot and are not perturbed by concurrent execution.

This buys three properties the spec calls out: speculation cannot contaminate verified state; the agent can hold multiple contradictory hypothetical futures at once (one branch projecting success, another failure) without internal paradox; and there is a structural basis for a containment layer.

Each branch is multi-dimensional. A branch does not carry a single score. Per the disclosure, each branch encodes a speculative mutation sequence, a projected outcome (the expected terminal state if executed), an affective reinforcement tag (emotional valence relative to the agent's disposition and intent), a trust slope projection (the hypothetical trust trajectory that would result), a policy compatibility flag, and a

classification label. This is what "multi-dimensional trajectory" means concretely: the future is evaluated along outcome, trust continuity, policy, and affective dimensions at once.

The six-phase forecasting execution cycle. This is the heart of the inventive step. It is described as a synchronous component of the deliberation pipeline, invoked at each cognitive decision point (act, delegate, or defer), not a background batch job:

1. **Initialization.** Read current verified state (intent, context, memory, affective, integrity, policy fields) and construct or refresh a planning graph rooted at that state, generating an initial branch set.
2. **Speculative mutation simulation.** For each branch, apply its hypothetical mutations to a sandboxed copy of state, not to verified memory, and compute the projected outcome, including projected environmental responses and secondary effects on the agent's own affective and integrity fields. The spec states this simulation is deterministic: same input state and mutation sequence yield the same projected outcome.
3. **Slope projection and validation.** For each simulated branch, compute a hypothetical trust-continuity hash and check whether promoting the branch would preserve the agent's trust-slope continuity or introduce a discontinuity; discontinuous branches are flagged ineligible. This acts as a prospective filter so the promotion pathway never even receives a branch that would fail validation.
4. **Policy compatibility check.** Evaluate each slope-eligible branch against current policy; branches containing policy-excluded mutations are reclassified or pruned.
5. **Emotional reinforcement tagging.** Tag each surviving branch with affective valence, influencing its priority.
6. **Branch marking and pruning.** Classify each branch (see taxonomy below) and schedule non-viable ones for removal.

The branch taxonomy. After the cycle, each branch is labeled: *eligible* (passed slope and policy, positively or neutrally reinforced, ranked by a composite score and a candidate for promotion); *introspective* (structurally viable but affectively aversive, retained for self-examination rather than promoted); *delegable* (viable but better suited to a child agent); or *pruned* (failed a check or superseded, retained briefly with a rejection annotation, then removed). Classification is not permanent; the cycle re-evaluates labels as state evolves.

The containment layer and the promotion interface. Nothing crosses from speculative to verified except through the promotion interface, which runs the full governance evaluation (policy, trust-slope, integrity, capability) and either admits the branch as a committed mutation or returns it to the speculative domain with a rejection annotation. Every planning-graph element carries an immutable speculative marker that only the promotion interface can strip. The spec frames the failure of this boundary as a "delusion boundary" condition: the pathological state where the agent can no longer tell what it projected from what actually happened. Designing the boundary as an enforced invariant is what prevents that.

Forecasting as input to confidence. The cycle also feeds a confidence governor. When the forecasting engine finds that every branch is pruned, introspective, or ineligible, so no eligible path to the intent exists, it transmits a negative viability signal that reduces confidence and moves the agent into a non-executing cognitive mode. In that mode the agent keeps forecasting (broader search, longer horizon, relaxed biases), generates inquiries to humans or other sources, and explores delegation, rather than acting with no viable plan. This is the mechanism that makes the agent pause instead of pushing forward on a doomed action.

How to Approach the Build

You are implementing this yourself. A workable order:

1. **Split state into two domains first.** Before any forecasting logic, establish verified execution memory and a separate speculative domain, with the only write path from speculative to verified being a single promotion function. Get this boundary right before anything else; it is the invariant everything else depends on.
2. **Make branches carry the full descriptor.** Model a branch as the mutation sequence plus projected outcome, trust-slope projection, policy flag, affective tag, and classification. Resist collapsing this to one number early; the multi-dimensionality is the point.
3. **Write a deterministic sandbox simulator.** Given a state snapshot and a mutation sequence, produce a reproducible projected outcome, including projected effects on the agent's own internal fields. Illustrative interface sketch (labeled illustrative; adapt to your stack):

```
# illustrative only, faithful to the disclosed cycle
snapshot = verified_state.read_root()          # phase 1
for branch in graph.branches:
    projected = sandbox.simulate(snapshot, branch.mutations)   # phase 2,
    branch.slope_ok = slope.continuous(snapshot, projected)    # phase 3
    branch.policy_ok = policy.admits(branch.mutations)         # phase 4
    branch.affect   = affect.tag(projected)                    # phase 5
    branch.label    = classify(branch)                         # phase 6
```

4. **Implement slope projection as a prospective filter.** Compute the hypothetical continuity of each branch and drop discontinuous ones before they reach promotion, so the governance path only ever sees viable candidates.
5. **Add the promotion interface as the sole gate.** It runs full governance validation and is the only code that clears the speculative marker. Never let any other path write speculative content into verified memory or lineage.

6. **Record forecasting in lineage as cognitive events.** Per the disclosure, the speculative content of branches is not committed to lineage, but lineage may record the forecasting process itself (graph creation, evaluation, pruning) as cognitive events. Log those so the agent's behavioral trajectory is reconstructible.
7. **Wire the negative-viability signal to a pause.** When no branch is eligible, reduce confidence and enter the non-executing mode (keep forecasting, ask, or delegate) instead of executing.

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 every component, including the substrate-level separation the invariants assume. The specification describes the design; it does not report benchmarks, latency figures, or accuracy numbers, and nothing here is a shipping, production-proven product. The determinism and separation properties are properties of the design; they hold only if you actually build the boundary as an enforced invariant rather than a convention, which the spec is explicit is the hard part.

The approach also does not turn a bad model into a good one. The projected outcomes are only as sound as your simulator; if your sandbox mis-models the environment, the forecasts will be confidently wrong. And it is aimed at agents with structured, mutable internal state and a governance pipeline. If your agent is a single stateless prompt with no verified-state concept, there is nothing yet for the containment boundary to separate, and you would build that substrate first.

Disclosure Scope

The forecasting execution cycle, planning-graph separation, six-phase execution cycle, branch taxonomy, containment layer, and forecasting-to-confidence coupling described here are disclosed in United States Patent Application 19/647,395. This guide is educational: it teaches how to approach building such an architecture from that public disclosure. It is not a warranty, not an offer of software, and not a representation that any implementation is provided. Any system you build from these ideas is your own implementation, and you are responsible for its correctness, safety, and compliance.

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)