Nick Clark · Published April 25, 2026 · PDF

How to Make an AI Agent Plan Several Steps Ahead Before Acting

Most agents pick the next action and only discover the consequences after they have already committed to it. This guide teaches an architecture that lets an agent project its own future state several steps ahead, evaluate candidate actions speculatively, and commit only the ones that survive. The approach described here is disclosed in United States Patent Application 19/647,395; it is an architecture you build, not a shipping library. Its home is the Forecasting Engine inventive step.

What You Are Building

You want an agent that does not lunge at the first plausible action. Instead, before it commits to anything, it should project where a sequence of candidate actions would take it, look several steps down each path, and only then decide what to actually do. The projections must be genuine dry runs: the agent reasons over hypothetical futures without any of them touching its real, committed state until it deliberately chooses one.

This is the problem behind search queries like "how do I make an agent think before it acts" or "how do I stop my agent from taking irreversible actions it regrets." If you are building a tool-using agent, a workflow orchestrator, or a delegating multi-agent

system, you have felt the failure mode: the model produces a confident next step, you execute it, and the downstream consequences only become visible once they are already real.

The architecture described here comes from the Forecasting Engine inventive step disclosed in United States Patent Application 19/647,395. It is a design you implement yourself, not a package you install.

Why the Obvious Approaches Fall Short

The common approaches are all real and all useful in their place, but each leaves the same structural gap.

Prompt the model to "think step by step." Chain-of-thought reasoning produces a plan in text, but the plan and the execution live in the same channel. There is no architectural boundary between "I am imagining doing this" and "I have done this." The model can talk itself into treating an imagined success as a real one.

Add an explicit planner that emits a sequence of steps. This helps, but a flat plan is committed as a whole. When step two of a five-step plan invalidates the assumption behind step four, a plan-then-execute pipeline has no built-in way to notice, because it never modeled the intermediate states the plan would pass through.

Use statistical tree search (for example, Monte Carlo Tree Search). Tree search does look ahead, and it is the right tool for games and other domains with a clean simulator and a scalar reward. It explores by sampling many random rollouts and scoring them statistically. For an agent whose "moves" are governance-constrained mutations to its own persistent state, random rollouts are the wrong shape: you want each projected step to be reproducible and to be checked against the agent's own policy and provenance constraints as it is generated, not averaged over noise afterward.

The gap common to all three: none of them maintains a hard, architectural separation between speculative reasoning and committed state, and none of them filters candidate futures against the agent's own constraints *before* those futures can become actions.

The Architecture

The disclosed design puts speculation in its own domain and gates the exit.

Two separate memory domains. The agent keeps its **verified execution memory** (the committed values of its fields, the lineage of validated changes, the results of executed operations) strictly separate from its **planning graphs** (the speculative structures where hypothetical futures live). Per the disclosure this is an architectural invariant, not a naming convention: no planning-graph branch can modify verified memory except by passing through a single governance-validated **promotion interface**. The separation is bidirectional. When the engine builds a planning graph it reads the current verified state as the graph's root node once, as a snapshot, and does not keep a live reference, so concurrent changes to verified state do not silently perturb an in-flight projection.

Two consequences fall out of this. Speculative reasoning cannot contaminate verified state: projecting a successful outcome does not make the agent believe it achieved that outcome. And the agent can hold multiple contradictory futures at once, one branch projecting success and another projecting failure, without contradiction, because neither is verified.

A planning graph of candidate futures. Each candidate action, and each sequence of them, is a branch. A branch represents a hypothetical mutation sequence. Branch depth is how many steps into the future the agent projects along that path before it stops exploring; branching factor is how many alternatives it opens at each step.

A forecasting engine with five components, running as a pipeline over the graph:

1. *Instantiation logic* creates new planning graphs from the current verified state, reading the intent field to know the objective, the context block to know the conditions, and the memory field to reuse prior outcomes.
2. *Affective prioritization* orders and weights branches by the agent's current disposition (for example, a risk-sensitive posture elevates conservative branches).
3. *Slope validation* checks each branch against the agent's trust-slope trajectory (below).
4. *Personality modulation* adjusts breadth, depth, risk profile, and temporal horizon of the speculation.
5. *Pruning manager* removes branches that are no longer viable under entropy thresholds, compute budgets, and expiration policies.

The six-phase forecasting cycle runs synchronously at each decision point, not as a background job:

1. **Initialization** reads current verified state and builds or refreshes the graph root; existing branches are re-checked for viability.
2. **Speculative mutation simulation** applies each branch's hypothetical mutations to a *sandboxed copy* of state, never to verified memory, and computes projected outcomes, including projected environmental responses and projected secondary effects. The disclosure states this simulation is deterministic (same input and mutation sequence yields the same projection) and, unlike Monte Carlo Tree Search, operates on defined structural fields and is constrained by policy and trust-slope continuity at every step rather than scored over random rollouts.
3. **Slope projection and validation** computes, for each branch, a hypothetical Derived Anchor Hash (DAH') representing the trust-slope state that would result if the branch executed, and checks whether it maintains lineage continuity.
4. **Policy compatibility check** rejects branches whose mutations are not admissible under current policy.

5. **Emotional reinforcement tagging** tags each surviving branch with how well its projected outcome aligns with the agent's disposition.
6. **Branch marking and pruning** classifies every branch.

Slope-constrained speculation is the key filter. The trust slope is the agent's cryptographic lineage trajectory. The disclosure makes slope eligibility a *hard* boundary, not a preference: a branch whose projected DAH' would break lineage continuity is slope-ineligible and can never reach the promotion interface. This filtering is prospective, so the governance pipeline never even receives a candidate that would fail validation, which concentrates compute on branches that have a real path to execution.

Four branch classifications determine what happens to each future: *eligible* (slope-valid, policy-compatible, positively reinforced; ranked by a composite score and available for promotion), *introspective* (viable but emotionally aversive; retained so the agent can examine its own aversions rather than acted on), *delegable* (better handed to a child agent), and *pruned* (failed a check or superseded). Classification is not permanent; the cycle re-evaluates it as state evolves, so an introspective branch can later become eligible.

A containment layer enforces all of this. Every element in a planning graph carries an immutable speculative marker that no in-graph operation can strip; only the promotion interface removes it, after governance validation, before writing to verified memory. Execution processes that query a field get the verified value, never a projected one. This is the disclosure's guard against the pathology of treating an imagined future as reality.

How to Approach the Build

The following ordered steps are how a developer would stand this up. The interface sketches are illustrative only and faithful to the disclosure; they are not a library.

1. **Split your state store in two.** Give the agent a verified store and a separate speculative store. Enforce that the only writer to the verified store is a single promotion function. If any other code path can write verified state, you do not yet have the architecture.
2. **Make speculation operate on a sandboxed copy.** Your simulation step must apply hypothetical mutations to a copy, compute a projected outcome, and return it without side effects. Keep it deterministic so the same branch reproduces the same projection.

```
# illustrative, not a package
simulate(branch, state_snapshot) -> ProjectedOutcome   # no writes to ver
```

3. **Represent futures as a branching graph, not a flat list.** Each branch is a mutation sequence with a depth (steps ahead) and children (alternatives). This is what lets you look several steps ahead along multiple paths at once.
4. **Implement your own slope/eligibility filter.** You need a reproducible way to ask "if this branch executed, would it keep the agent's committed lineage continuous and legal under policy?" and to answer it *before* promotion. Whatever your provenance mechanism is, compute the hypothetical post-branch state and validate it prospectively. Branches that fail are barred from promotion, though you may retain them for introspection.
5. **Order the survivors.** Rank eligible branches by a composite score (projected outcome quality, how far the branch advances the trajectory, projected integrity impact, disposition alignment, intent alignment). The top-ranked eligible branch is your leading candidate, still subject to the promotion gate.
6. **Gate the exit with one promotion interface.** The leading candidate passes through governance evaluation; on success, strip the speculative marker and write to verified memory; on failure, return it to the speculative domain with a rejection

annotation. There is no other path from speculative to verified.

7. **Control the horizon.** Expose branch depth and branching factor as tunable parameters and let them be modulated: under time pressure the disclosure compresses horizons (less depth, a narrower projection window) to prioritize near-term paths; a more exploratory disposition widens breadth and depth. Add a pruning manager that trims on entropy, compute budget, and expiration so the graph does not grow without bound.
8. **Loop it at every decision point.** Run the six-phase cycle synchronously each time the agent must act, delegate, or defer, re-classifying branches as state changes.

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 component against your own state model, policy engine, and provenance mechanism.

It is disclosed in a patent filing. It has not been presented here as a benchmarked or production-proven system, and this guide states no latency, accuracy, or throughput numbers, because the disclosure does not and neither should you.

The design assumes your agent has structured, persistent state you can snapshot and validate, and a governance or provenance notion you can project forward (the trust-slope check). If your agent is a single stateless prompt with no committed state and no policy layer, most of the value here (the verified/speculative split, prospective slope filtering, the promotion gate) has nothing to attach to. The deterministic simulation also depends on your ability to model projected outcomes reproducibly; where your environment is genuinely stochastic and unmodelable, a statistical method may fit that part better. Finally, looking further ahead costs compute; depth, breadth, and pruning are tradeoffs you tune, not free guarantees.

Disclosure Scope

The self-forecasting architecture described in this guide, including the separation of speculative planning graphs from verified execution memory, the six-phase forecasting execution cycle, slope-constrained speculative simulation, branch classification, and the containment layer, is disclosed in United States Patent Application 19/647,395. This guide is educational: it explains an architectural approach so that a skilled developer can implement it independently. It is not a warranty, not an offer of software, and not a representation that any product implementing this architecture exists or performs to any particular standard.

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)