

Steering the Talking Probability Machine

An Illustrated Primer
on Language Models,
Feedback, Agents, and
Stochastic Control

JUNE 2026 · AI, JOBS & THE FUTURE OF WORK

AUTHOR

Maximiliano Armesto

CTO at Taller
maximiliano.armesto@tallertechnologies.com

Jan Rosen

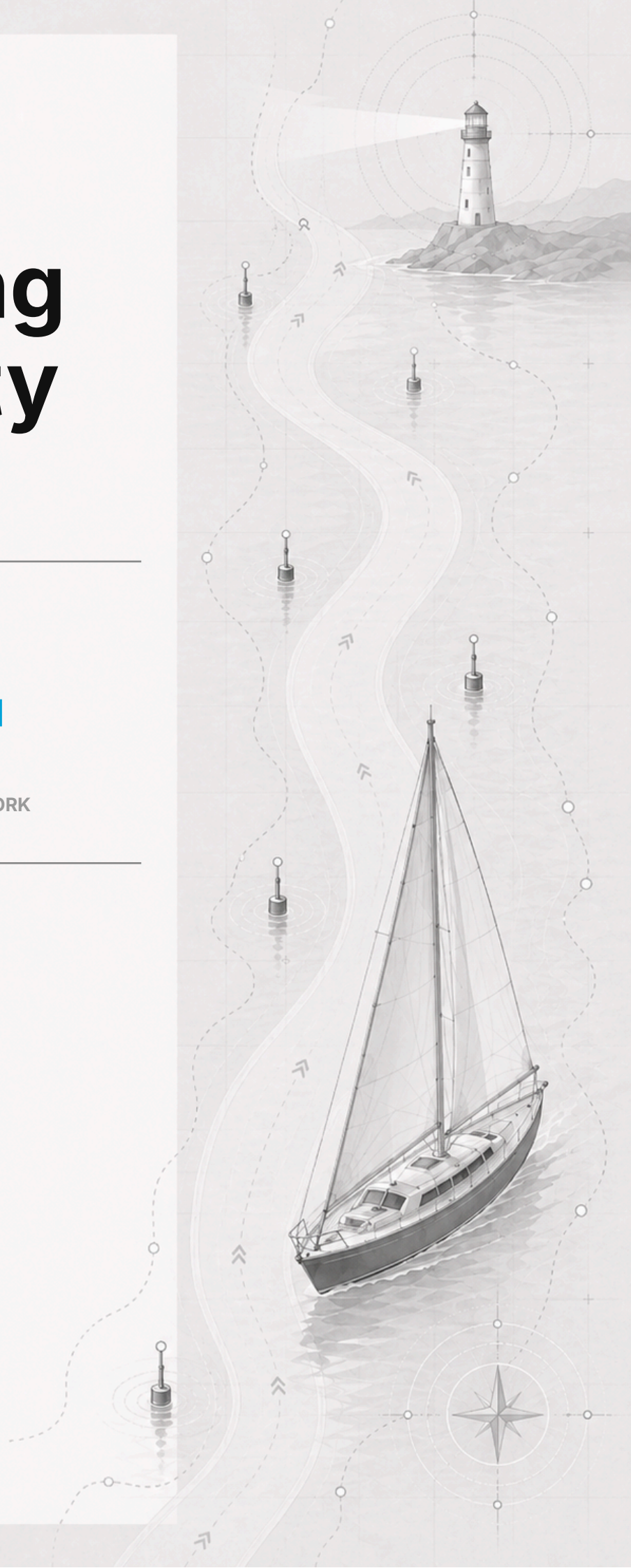
CITO at Taller
jan.rosen@tallertechnologies.com

Martin Gonella

Director of Transformation at Taller
martin.gonella@tallertechnologies.net

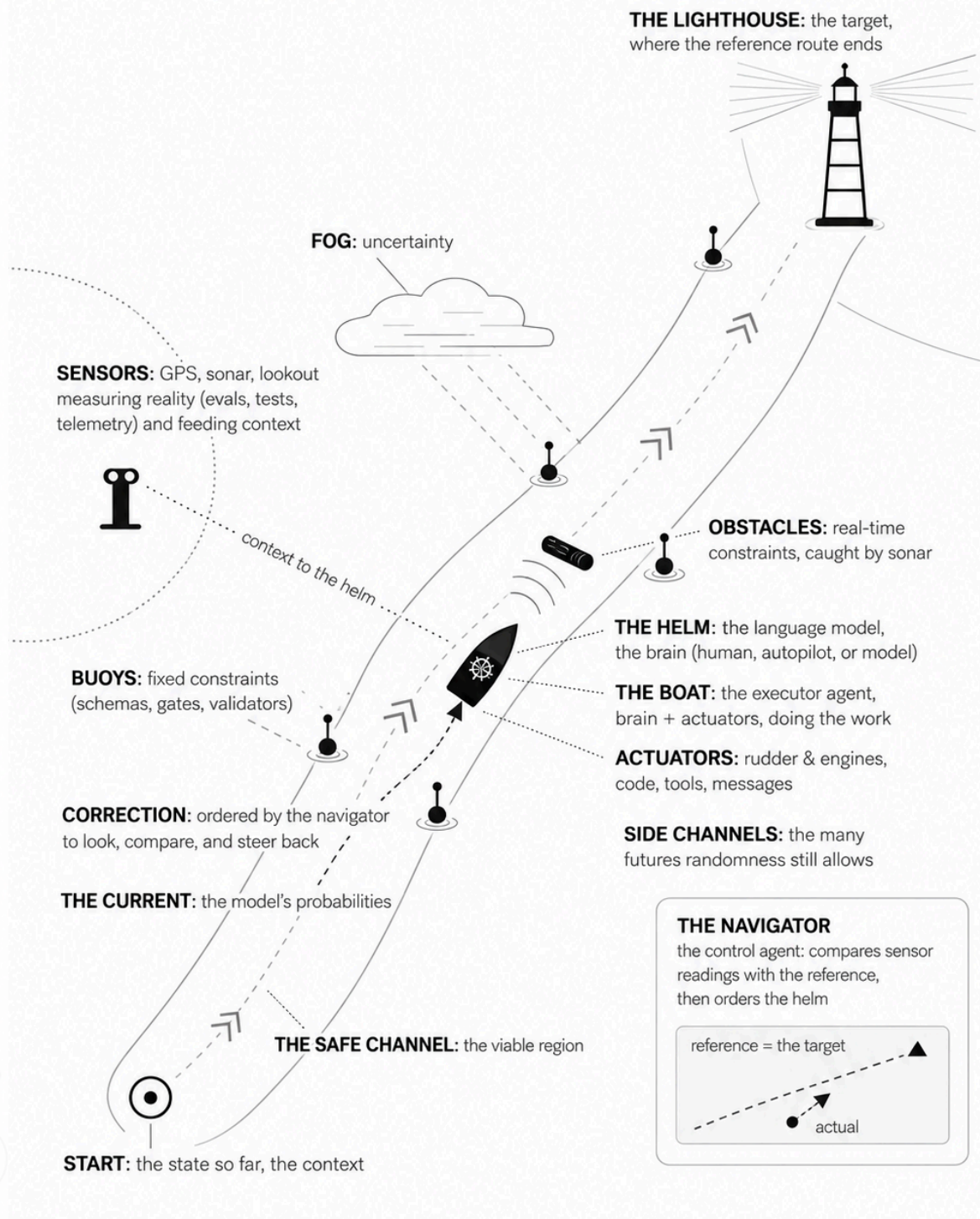
Christophe Kolb

CEO at Taller
christophe.kolb@tallertechnologies.com



Steering the Talking Probability Machine

A chart of the voyage: every object on the water is a control idea



You steer the voyage by shaping the odds.

The working sentence

A language-model agent is a probability machine whose future path can be steered through controls that change odds, sensors that reveal drift, and correction loops that compare the voyage against its reference and keep it inside a safe channel.

Abstract

This primer builds one bridge, plank by plank: A language model is a stochastic dynamical system, and reliable AI engineering is a control problem. A brain at the helm chooses each next move from a probability field. Sensors measure reality and feed the helm its context. A control agent compares those measurements with the reference, the planned trajectory, and orders corrections. Seventeen short sections walk from next-token probabilities to reachability, feedback, entropy allocation, observability, damping, hidden burden, and the operating model that keeps deployed systems inside a viable region.

Contents

1. One river, many futures
2. The box that weights continuations
3. Prompts as control inputs
4. The map of reachable futures
5. Feedback: look, compare, correct
6. Agents make voyages
7. Shaping the cloud
8. The entropy window
9. Sensors and observability
10. Leverage inside attention
11. When mistakes echo
12. Tools as mode switches
13. Burden as a hidden cost
14. The worry-meter
15. The control room
16. Try it yourself
17. Field guide and mathematical core
18. Further reading

How to read this primer:

Keep one image in mind: a small boat on a branching river. The language model supplies currents of possible continuations. Prompts, schemas, tools, memory, and review gates act as steering surfaces. Control theory supplies the craft of staying in the channel.

1. One River, Many Futures

The central metaphor: A language-model agent is a boat in a current, with the model at the helm.

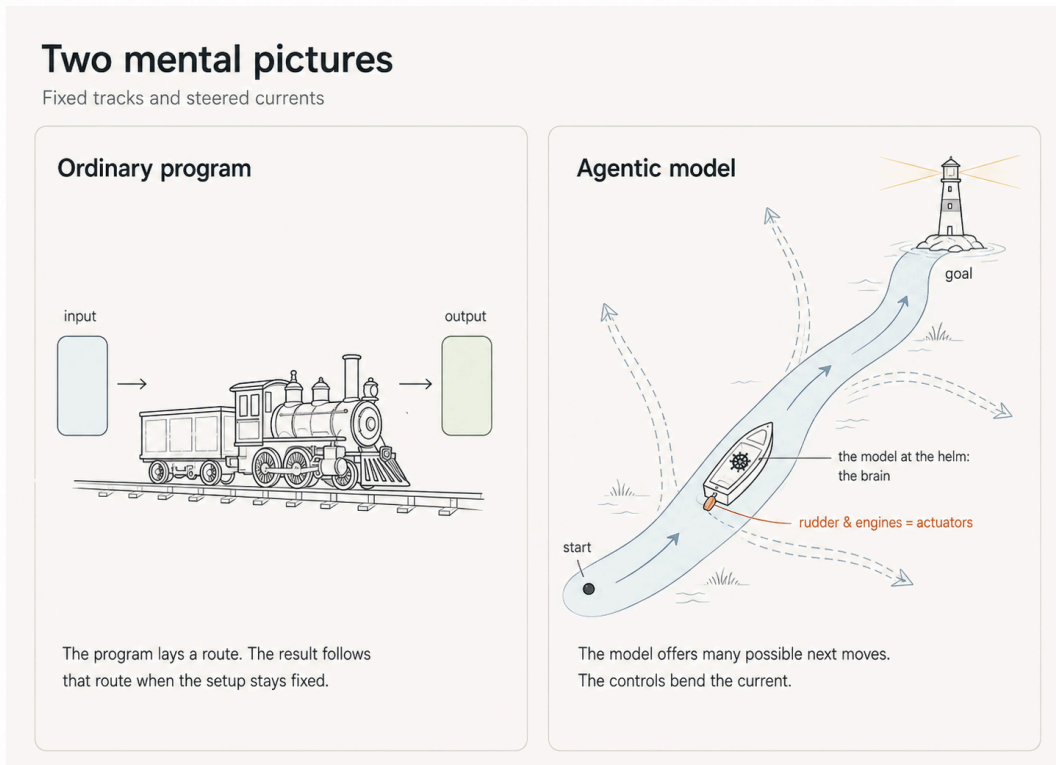


Figure 1. Ordinary programs feel like tracks. Language models feel like steered currents.

Many ordinary programs feel like train tracks. A route is laid down in advance. Given a fixed input and fixed environment, the result usually follows the planned rail.

A language-model system feels closer to a boat in moving water. At the helm sits a brain: a human pilot, an autopilot, or a model. The boat has actuators (rudder and engines), sensors, a chart that carries the reference route, and a destination. The river has currents, eddies, fog, and wind. A prompt is an order to the helm; the brain turns the rudder. A system instruction sets a larger trim. Retrieval gives the chart its detail. Tools provide docks and machines along the river. Evals are the sensors: binoculars, depth gauges, GPS, sonar.

Core intuition:

Reliability means keeping the voyage inside the safe channel while preserving enough motion to search, explain, compose, and adapt.

The language-model question changes from a single sentence to a journey. Which futures become likely? Which futures become rare? Which sensors reveal drift early? Which correction loops bring the boat back cheaply?

A first-principles ladder

Stochastic control begins with a few ordinary ideas: state, action, uncertainty, observation, cost, and correction. A language model gives each idea a new costume.

- The state is the conversation and surrounding context.
- The action is a prompt, tool choice, schema, retrieval write, memory edit, or escalation.
- The uncertainty is the distribution over continuations and actions.
- The observations come from tests, validators, traces, reviewers, and telemetry.
- The cost includes wrong answers, unsafe tool calls, hidden cleanup, and delay.

Once those pieces are visible, LLM engineering becomes a control problem: Sense where the system is, compare it with the desired channel, choose a steering input, and measure again.

2. The Box That Weights Continuations

The model contains a rule for weighting many continuations.



Figure 2. At generation time, the model builds a weighted field over possible next tokens.

At each step of generation, a model turns the current history into a probability distribution over possible next tokens. The token with the highest weight may appear. A lower-weight token may also appear when sampling allows exploration.

The important unit is the distribution. A sentence grows through repeated draws from a changing distribution. Each new token becomes part of the history. The next distribution is then recomputed from the enlarged history.

The simplest formula

next token $\sim P(\text{token} \mid \text{history})$

The model is a stochastic dynamical system: A state evolves step by step, each step depends on the current state, and randomness shapes the transition.

The state is the story so far

The word state sounds abstract. In a conversation, it means where the story currently stands; in a language model, the equivalent is user request, system instruction, retrieved passages, previous model tokens, tool results, memory entries, and interface constraints. The model continues from that whole situation.

A tiny change in state can change the distribution. The phrase “the bank was” can point toward a riverbank in one history and a financial institution in another. The symbol stays the same. The surrounding state changes the odds.

3. Prompts as Control Inputs

Prompting works by reshaping probabilities.

Prompts tilt the landscape

A control input reshapes which continuation rolls downhill

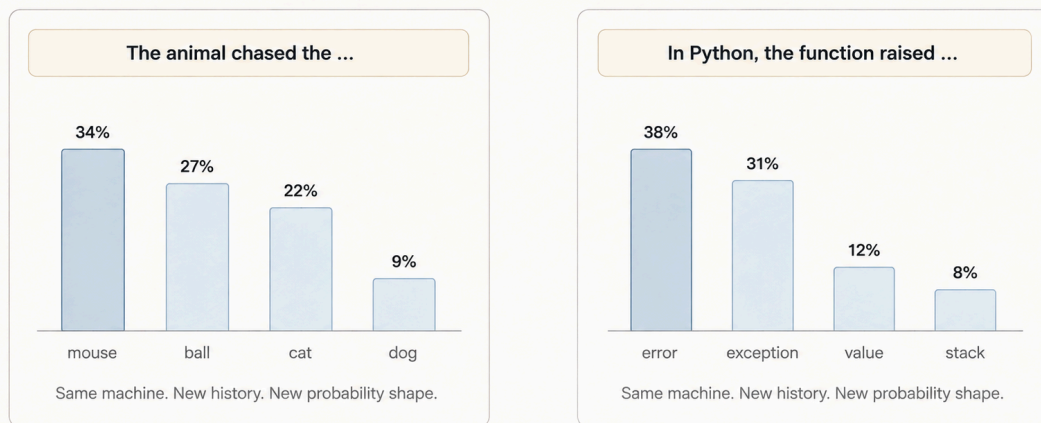


Figure 3. Two prompts send the same model toward different probability landscapes.

A prompt is a control input because it enters the state and reshapes the next distribution. It can make a class of continuations more likely, suppress other continuations, constrain form, select tools, invite deliberation, or call for a check.

Prompting has the flavor of steering. A few words can move probability mass. Longer instructions supply a larger control budget. A schema can carve a narrow channel. A verifier can send a correction signal after the model has drifted.

Control vocabulary

- State x_0 : where the conversation is now.
- Control u : the nudge supplied by prompt, schema, tool gate, or memory edit.
- Output y : the next token, answer, action, or artifact.

This view also explains prompt brittleness. A small rudder movement may work in calm water and fail in a strong current. The same instruction can steer well from one state and weakly from another.

Control budget

A one-word hint, a three-sentence instruction, a typed JSON schema, and a multi-step review policy supply very different amounts of leverage. Control budget includes length, precision, placement, retrieval quality, tool gating, compute, and review effort. Reliability depends on using the smallest effective budget for the local risk.

4. The Map of Reachable Futures

Reachability asks which futures your available controls can reach.

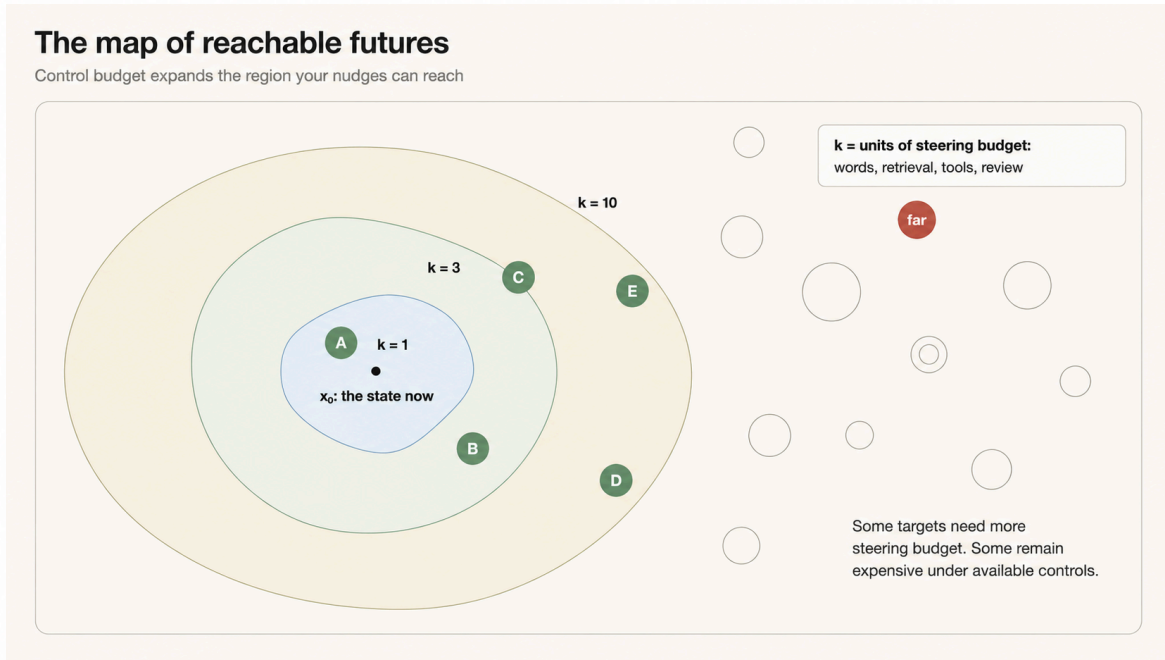


Figure 4. More steering budget expands the reachable region, while distant targets remain costly.

Imagine every possible output as a point on a map. Starting from state x_0 , a control input of length one reaches a small region. A longer sequence reaches a larger region. A richer controller, with retrieval, checks, and tool policies, reaches an even more useful region.

The reachable set is the region your controls can steer toward under a given budget. In symbols, $R^k_y(x_0)$ means the outputs y reachable from the starting state x_0 using up to k units of control. In plain speech: What can this system get to from here, using the knobs available, at this cost?

Reachability question:

Can the available controls make the desired target likely enough, soon enough, with acceptable repair cost?

This question guards against wishful engineering. Some targets require a longer prompt. Some require retrieval. Some require a tool. Some require a human gate. Some require a different model or a redesigned task. Reachability invites measurement instead of hope.

From target tokens to target journeys

At token scale, a target may be a particular next word. At task scale, the target is a finished artifact, a safe tool trace, or a correct decision with a clean audit trail. The control idea stays the same: Define the target set, define the controls, measure the reachable region, and choose a policy that keeps the system in viable territory.

5. Feedback: Look, Compare, Correct

A controller is a repeated comparison between the actual path and the desired path.

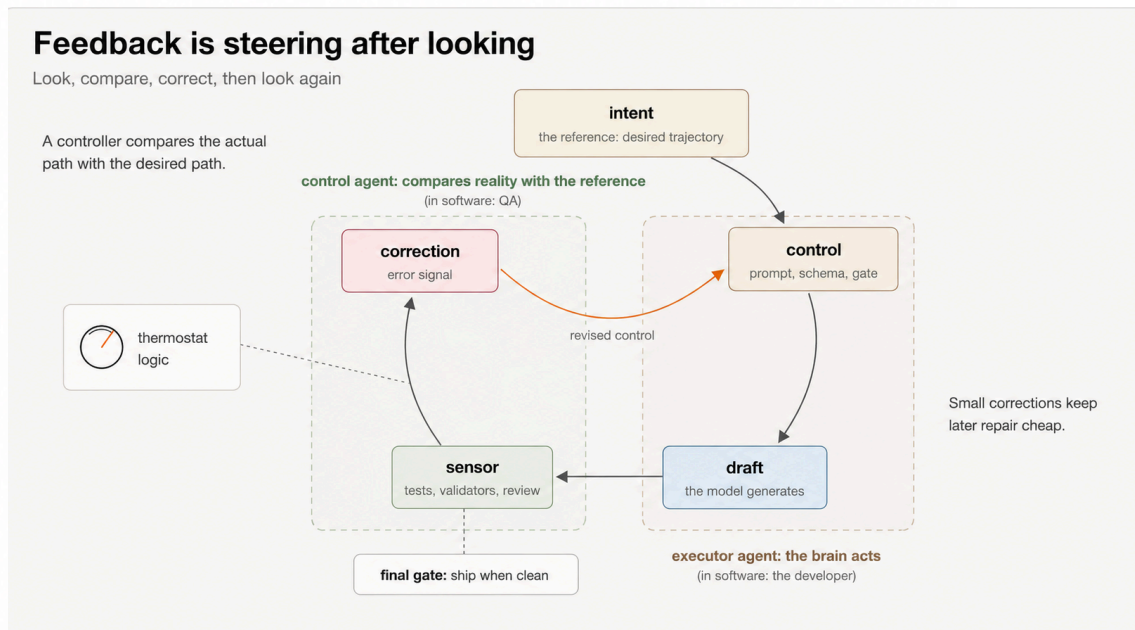


Figure 5. Closed-loop AI systems steer by sensing, comparing, correcting, and measuring again.

Open-loop prompting sends one instruction and accepts the result. Closed-loop prompting measures the result, compares it with intent, and sends a correction. That loop can be as simple as a user saying “revise the answer” or as structured as tests, validators, reward models, trace analyzers, and deployment gates.

In control terms: Sensors measure reality. The control agent compares those measurements with the reference, the planned trajectory. The executor agent, the brain plus its actuators, carries out the ordered correction. In software delivery, the executor writes the work and the control agent plays QA. The chart is where you see the target. The target itself is the reference: the trajectory the loop keeps comparing against.

A code assistant illustrates the idea. Ask for code, run tests, paste failures back, revise, run tests again. The model remains stochastic. The surrounding loop becomes corrective. The reliability comes from sensing and repair as much as from the raw generation step.

Control loop

intent → draft → check → error signal → revised control → new draft → final gate

Feedback turns mistakes into signals. A syntax error becomes a compiler message. A factual uncertainty becomes a request for evidence. A policy concern becomes escalation. A tool failure becomes a mode change. The loop converts drift into information.

Cheap correction beats late repair

Early sensors matter because error can spread. A false premise caught in a draft is inexpensive. The same premise written into memory, emailed to a customer, and copied into a report becomes costly. Control design places sensors before expensive or irreversible actions.

6. Agents Make Voyages

The primitive is the trajectory: the whole path through state, action, observation, and correction.

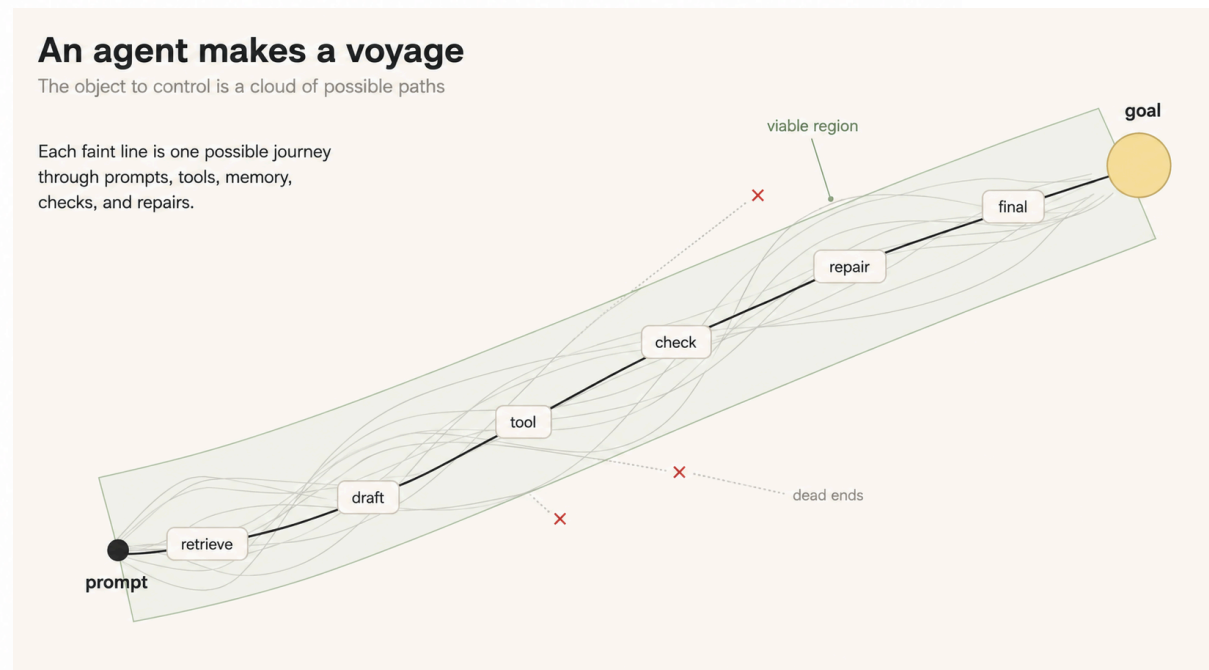


Figure 6. An agent's voyage passes through prompts, retrieval, tools, memory, checks, repair, and output.

A single answer is a point. An agent produces a path. The path may include retrieval, drafting, tool calls, code execution, memory writes, validation, repair, escalation, and final delivery. Each step changes the state for later steps.

That makes agent reliability a trajectory problem. A system that gives a correct final answer after ten hidden retries, three tool failures, two memory cleanups, and a human rescue carries a different risk profile from a system that reaches the same answer cleanly.

Trajectory view

A trajectory is the sequence of states, controls, outputs, observations, costs, and corrections that unfolds during the task.

The model is the brain of the executor agent. The rest of the system completes the plant: tools (the actuators), memory, retrieval, user context, artifacts, monitors, policies, and the people who review and operate it.

Path quality

A good path reaches the target, stays within constraints, leaves useful artifacts, records uncertainty, avoids hidden cleanup, and stops safely when confidence or authority runs out. This path view gives engineers more useful measurements than final-answer accuracy alone.



7. Shaping the Cloud

The engineering target is the distribution of possible journeys.

Four-line mathematical core

The whole bridge in compact form

1

`state_t = the story so far`

2

`token_{t+1} ~ P( · | state_t )`

3

`control_t = prompt, schema, tool gate, retrieval, memory, review`

4

`trajectory = state → control → output → observation → correction → ...`

Control shapes the probability of complete paths:
common good paths, rare costly paths, early visible drift.

Figure 7. The bridge in four lines: state, probability, control, trajectory.

For a language-model system, the future is a cloud of paths. Some paths reach the goal cleanly. Some wander and recover. Some drift into high-cost territory. The controller shapes the cloud: Good paths become common, risky paths become rare, and repair happens early.

This is stochastic control in everyday dress. The system evolves through random transitions. The controller chooses inputs based on observations. The design has a cost function: accuracy, safety, usefulness, burden, latency, and reversibility.

The control objective

- Raise the probability of useful trajectories.
- Lower the probability of costly trajectories.
- Shorten the time between drift and correction.

This objective is more realistic than chasing a single perfect output. It also scales. The same logic applies to one response, a tool-using assistant, a research agent, a customer-service workflow, and a robotic system that uses language to plan.

Viable region

The safe channel is a viable region: a set of states and paths where the system remains useful, bounded, and observable. A controller maintains viability by enforcing constraints, applying correction, and routing high-risk cases to stronger controls.



8. The Entropy Window

Uncertainty becomes useful when it is placed inside boundaries.

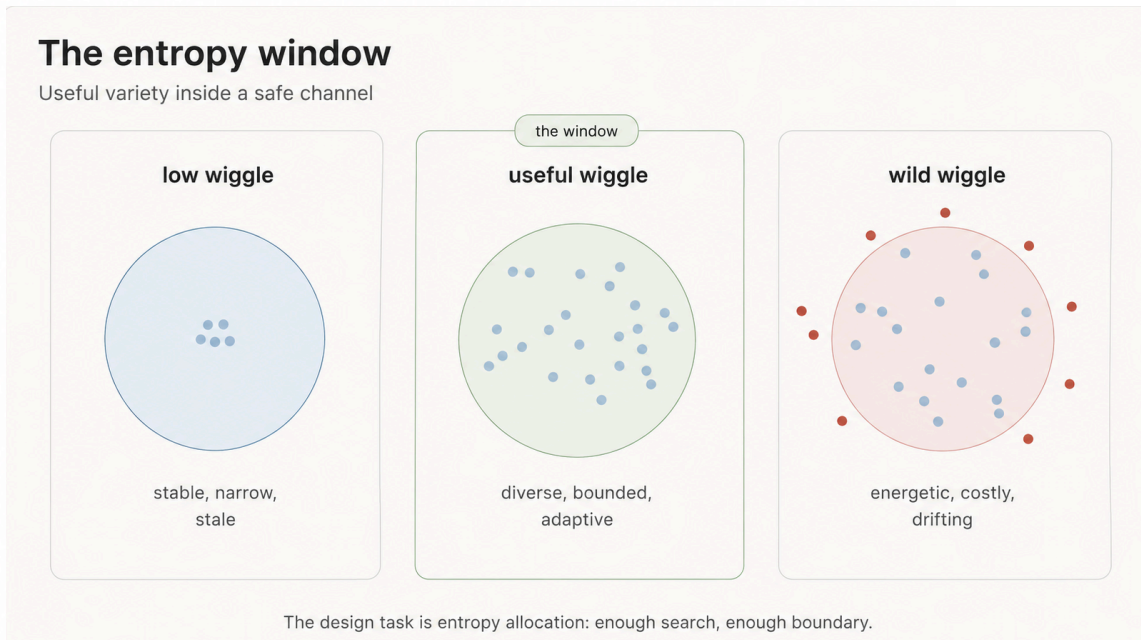


Figure 8. Low, useful, and wild variation show why reliability requires entropy allocation.

A language model is valuable because it searches a huge space of possible meanings, plans, phrases, analogies, and explanations. Too little variation yields stale repetition. Too much variation yields drift. The practical zone sits between those extremes: enough stochasticity for search, enough constraint for safety.

Think of a garden, where biodiversity encourages growth but weeds and invasive species prevent it. Useful variety belongs inside the fence: alternative phrasings, creative examples, candidate plans, and exploratory drafts. High-cost randomness belongs outside: hallucinated authority, tool misuse, prompt injection, privacy leakage, and uncontrolled memory writes.

Entropy allocation

- Put variation where it helps search.
- Put fences where variation creates cost.
- Match randomness to task risk.

Temperature, top-p sampling, schemas, retrieval grounding, system instructions, validators, and tool gates all allocate entropy. The best setting depends on the task: Brainstorming, summarization, coding, legal review, medical triage, data entry, or tool execution each calls for a different window.

The useful middle

A locked-down assistant can feel safe while losing the ability to explore. A wildly creative assistant can feel brilliant while shedding reliability. Controlled entropy gives the system room to search inside a bounded channel.

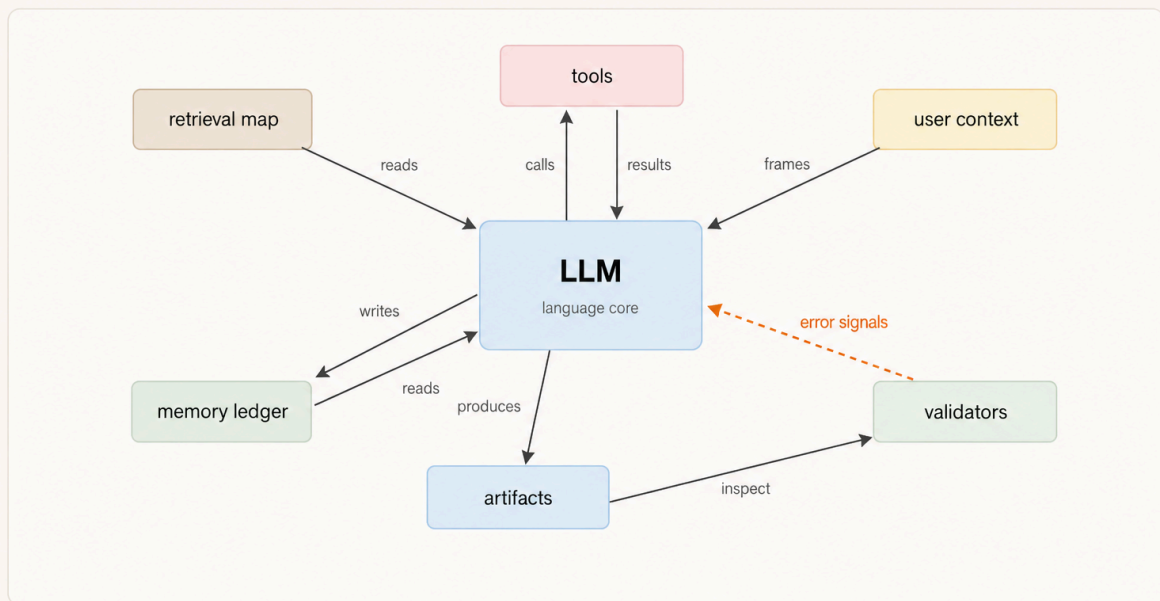


9. Sensors and Observability

A system can correct only what it can see.

The agent is larger than the model

The plant includes tools, memory, lasers, artifacts, and context



When a model can read, write, call tools, and update state, the surrounding system becomes part of the machine.

Figure 9. The controlled system includes the model and its surrounding workshop.

Observability asks whether the controller can infer the important state of the system from available signals. In language systems, the hidden state includes uncertainty, ambiguity, prompt-injection pressure, tool side effects, stale memory, and emerging drift.

Sensors make those hidden conditions visible. A sensor might be a unit test, a verifier, a fact checker, a schema validator, a trace viewer, a human review queue, an incident report, a drift monitor, or a burden dashboard.

Sensors do double duty. They feed the control agent, which compares reality with the reference. They also feed the helm: The brain acts differently depending on the context the sensors deliver, which is exactly what retrieval does for a language model. Some sensors watch the boat; others watch the water ahead. Obstacles arrive in real time: a changed requirement, an injected instruction, a failing dependency. The trajectory must bend around what the sensors catch.

Sensor design rule:

Place sensors where drift first becomes visible, especially before memory writes, external messages, payments, database mutations, and other high-authority actions.actions.

Good sensors are specific, cheap, and close to the failure mode. A compiler is a strong sensor for syntax. A retrieval citation check is a useful sensor for grounded summaries. A policy classifier can be a first-pass sensor for escalation. Human review remains valuable for ambiguous, high-impact cases.

Observability gaps

An observability gap appears when the system succeeds according to visible metrics while hidden costs accumulate. Examples include silent human cleanup, repeated retries, unresolved ambiguity, memory pollution, and user confusion. A mature AI system measures these hidden costs directly.

10. Leverage Inside Attention

Prompt tokens act through learned geometry.

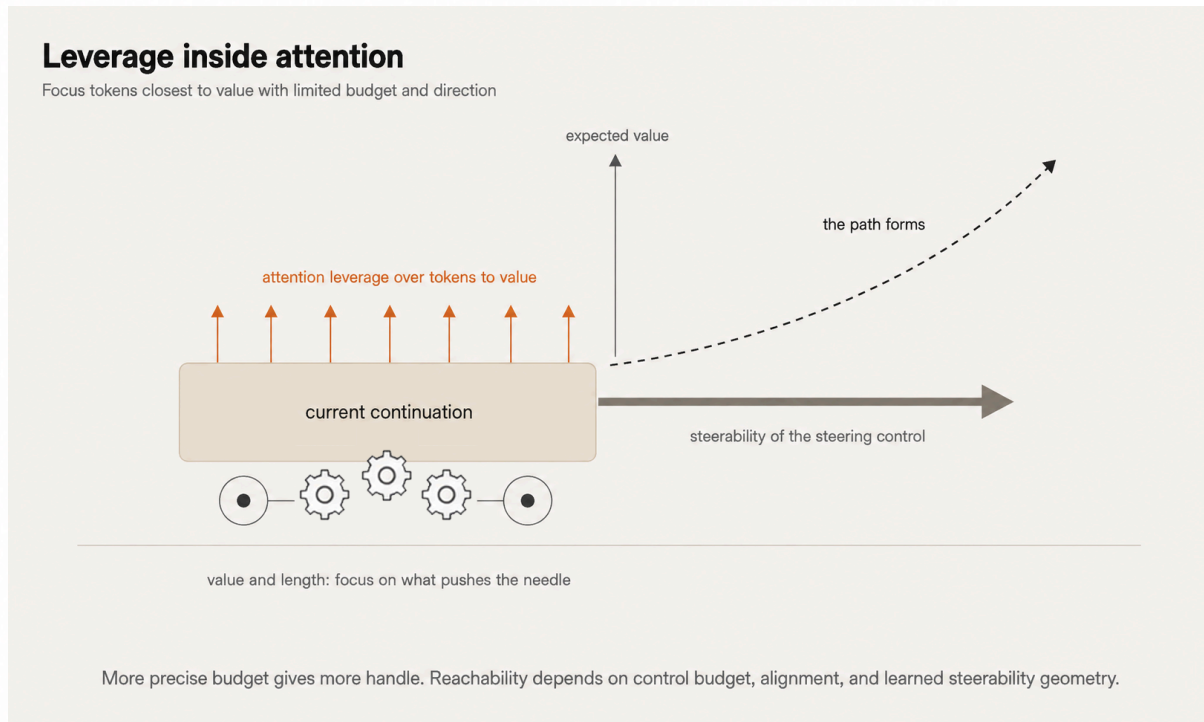


Figure 10. Prompt length, alignment, and learned internal gears determine steering leverage.

Inside a transformer, self-attention mixes information from the current context. Words become vectors. Those vectors create queries, keys, and values. Attention computes weighted combinations that influence later layers and the next-token distribution.

A prompt supplies extra vectors. Those vectors can attract attention, shift weighted averages, introduce constraints, and change which continuations become likely. A short prompt supplies a few handles. A long prompt supplies more handles. The handles also need useful direction. Strength without alignment wastes force.

Reachability as leverage

A strong current in the existing context can overwhelm a short control input. More prompt tokens, better placement, retrieval, or a different controller can increase leverage.

The cart provides an analogy. More handles means the cart can be more easily pulled in the right direction.

Practical lesson

Prompt engineering succeeds when the control input is placed where the model can use it, phrased in the geometry of the task, supported by the right context, and reinforced by feedback. More words alone can add clutter. Better leverage comes from state-aware controls.

11. When Mistakes Echo

Damping asks whether error shrinks or amplifies as the loop runs.

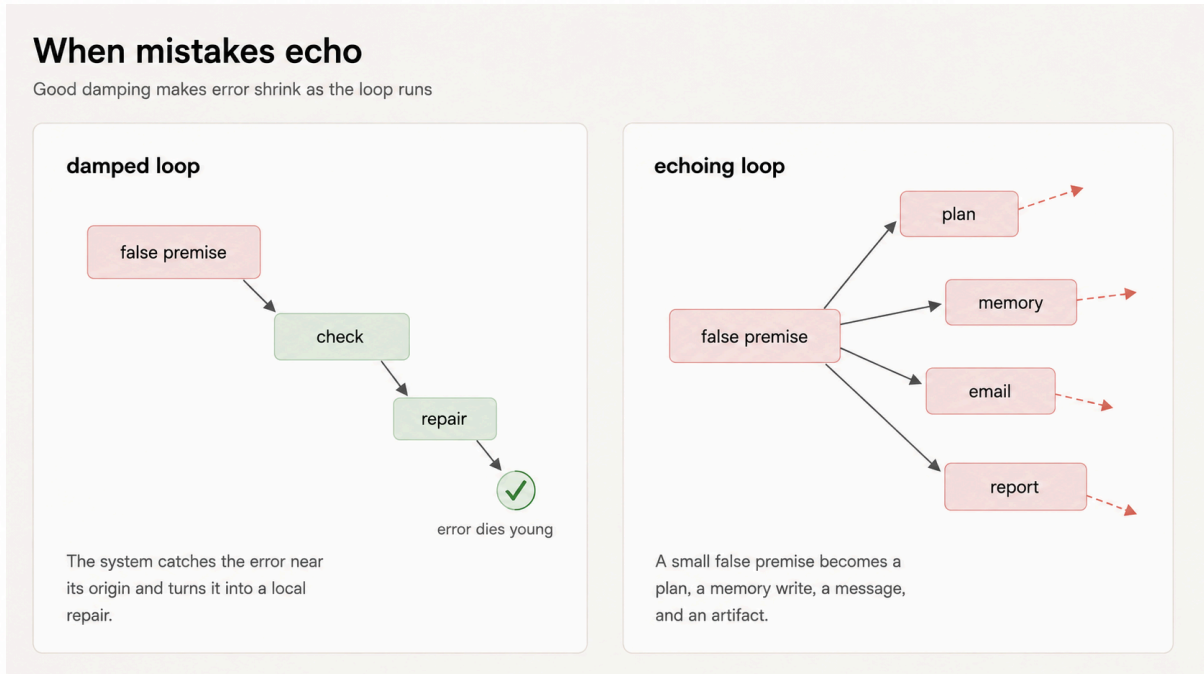


Figure 11. A damped loop catches a false premise locally; an echoing loop spreads it through artifacts.

Every model can produce an error. The control question is whether the error dies young or echoes. A damped system turns an error into a local correction. An under-damped system copies the error into memory, plans, tool calls, messages, and business decisions.

Damping comes from validation, reversible steps, isolation, rate limits, memory hygiene, and escalation. It also comes from refusing to let uncertainty masquerade as authority. A system with good damping may still make mistakes, yet those mistakes stay small and visible.

Damping metric

Track the fraction of errors caught at draft time, before tool use, before memory writes, before external communication, and before irreversible action.

Error amplification is especially dangerous in agents because actions create new state. A poor summary can poison memory. A bad plan can call a tool. A faulty tool result can become evidence for the next step. Damping interrupts that chain.

Localize uncertainty

Healthy agents keep uncertainty attached to its source. They mark claims that need verification, separate speculation from evidence, isolate tool results, and clear stale memory. Local uncertainty is manageable. Disguised certainty spreads.



12. Tools as Mode Switches

Tool calls change the dynamics of the system.

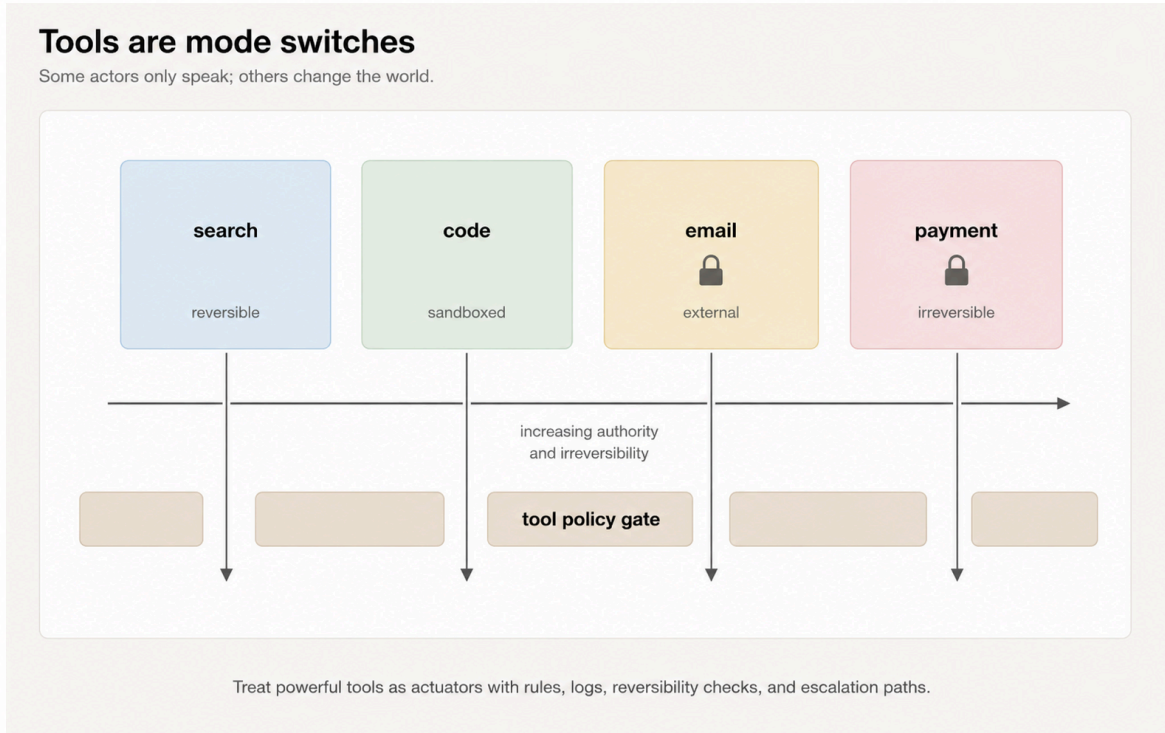


Figure 12. Tool use creates discrete modes with different authority and reversibility.

Text generation is one mode. Search is another. Code execution is another. Email, calendar writes, payments, database updates, and robot motion are higher-authority modes. A tool call changes the state transition law of the system.

This makes tool-using agents hybrid systems: Continuous language generation interleaves with discrete mode switches. Each mode needs rules. What inputs are allowed? What logs are kept? Which actions require confirmation? Which operations are reversible? Which sensors fire after the call?

Mode-switch rule:

Higher authority requires stronger sensors, tighter gates, clearer logs, and safer rollback paths.

A safe architecture treats tool access as an actuator design problem. Low-risk tools can be broad and fast. High-risk tools need narrow schemas, approval gates, sandboxing, and post-action checks. The language model proposes; the controller authorizes.

Prompt injection as hostile control

Prompt injection can be understood as an outside signal trying to seize the controller. The defense is architectural: Isolate instructions by authority, restrict tool scope, sanitize retrieved content, log mode switches, and place independent sensors around high-risk transitions.

13. Burden as a Hidden Cost

Task success can hide the work moved elsewhere.

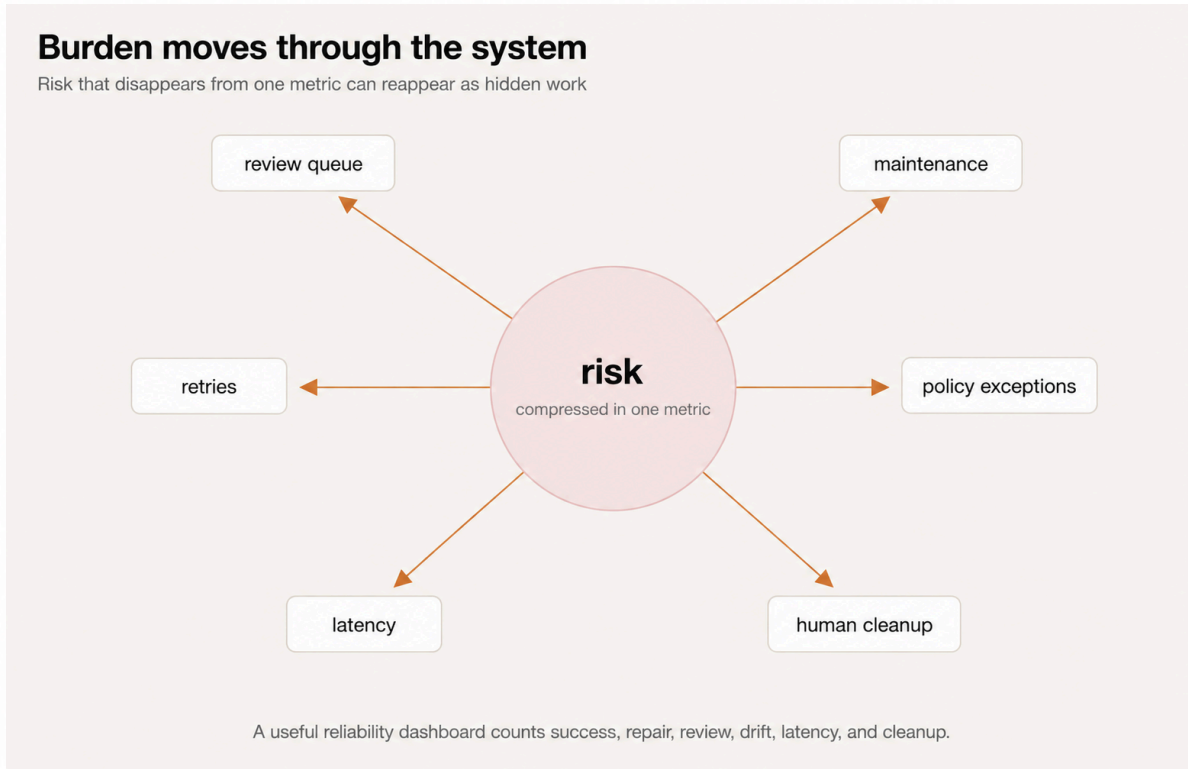


Figure 13. Risk compressed in one place can bulge out as review, retries, maintenance, exceptions, and latency.

A system can appear successful while shifting cost to users, reviewers, operators, and future maintenance. This hidden burden matters. It consumes time, attention, trust, and organizational capacity.

Burden shows up as retries, human corrections, validator failures, unexplained decisions, memory cleanup, policy exceptions, latency, and incident response. A reliability program that ignores burden rewards systems that look efficient by making other people absorb instability.

Burden audit

For every workflow, count final success together with retries, tool errors, human corrections, escalations, cleanup, delay, and user confusion.

Control theory makes burden visible by placing it inside the cost function. Accuracy remains important. So do safety, latency, reversibility, cleanup, and review load. A good controller minimizes total expected cost across the whole sociotechnical system.

The human layer is part of the loop

Human review is a powerful sensor and actuator, but it has capacity limits. The design goal is to route human attention to cases where judgment matters most, while automating low-risk checks and preserving auditability.



14. The Worry-Meter

Stability means the mess decreases as correction proceeds.

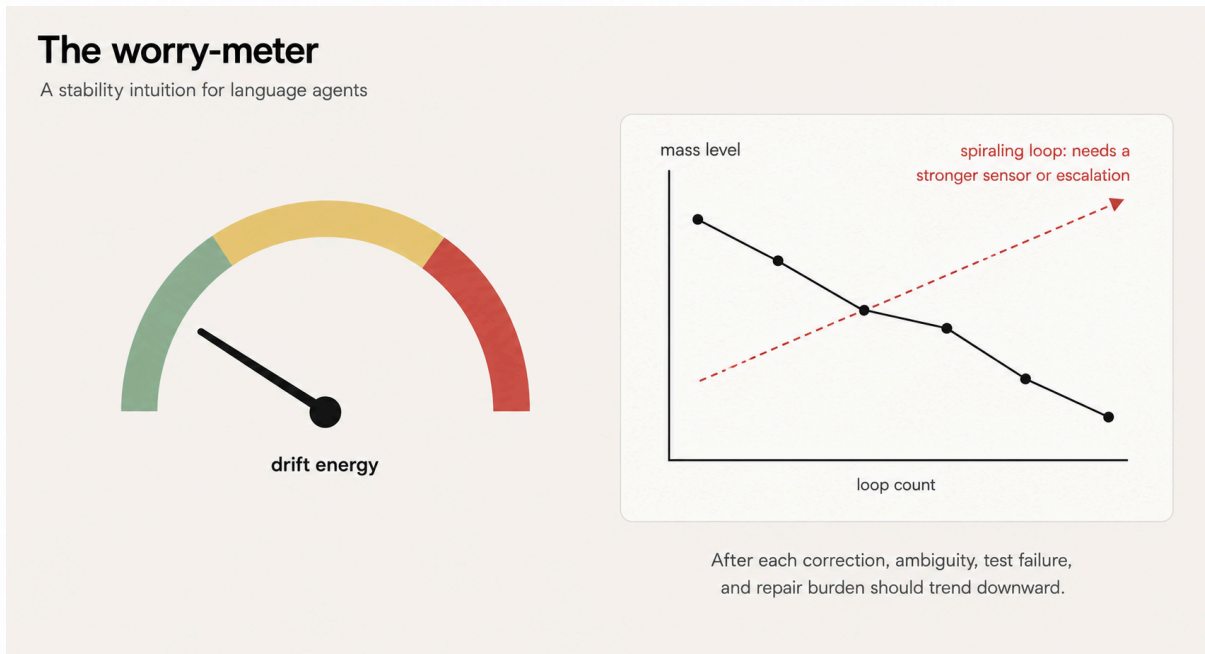


Figure 14. A Lyapunov-style worry-meter: After each loop, drift energy should trend downward.

Formal stability theory often uses a function that decreases as the system settles. For a primer, call it the worry-meter. If the system is correcting itself, some measure of mess should go down after each loop.

The mess can be hallucination risk, unresolved ambiguity, test failures, policy uncertainty, contradiction count, human repair burden, or tool error rate. The exact meter depends on the task. The principle is simple: Correction should reduce the quantity that represents danger or disorder.

Worry-meter examples

After each loop: fewer test failures, fewer unsupported claims, smaller ambiguity set, lower tool risk, less human repair burden, clearer evidence trail.

When the worry-meter rises after each retry, the system is spiraling. It needs a stronger sensor, a different controller, a clearer task decomposition, a safer mode, or escalation to a human owner.

Stability as design habit

Stability begins with asking what should decrease. For code, test failures. For factual answers, unsupported claims. For customer service, unresolved intent. For tool use, unchecked side effects. For memory systems, stale or contradictory entries.



15. The Control Room

The operating model is part of the controller.

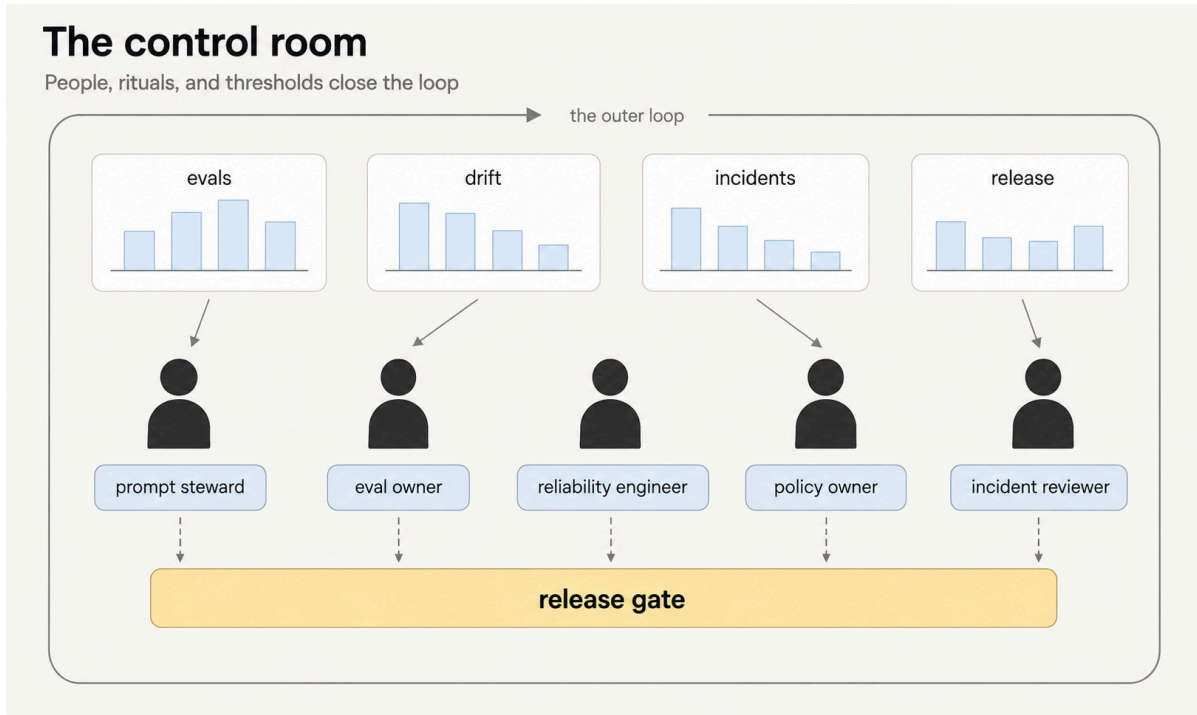


Figure 15. AI reliability depends on people, roles, dashboards, thresholds, and release gates.

In deployed AI, the controller includes code and people. Prompts are versioned. Evals have owners. Incidents are reviewed. Release gates define thresholds. Policy owners define authority. Reliability engineers monitor drift. Product teams decide acceptable tradeoffs.

This operating model closes the outer loop. A prompt change can create a new failure mode. A new tool can alter reachability. A policy update can change the viable region. A drift report can trigger rollback. Roles and rituals make these changes observable and governable.

Control-room checklist

Name the prompt owner, eval owner, tool owner, incident owner, policy owner, release gate, rollback path, and escalation rule.

The strongest systems treat prompts, schemas, evals, memories, and tool policies as controllers with versions, tests, approvals, and logs. Reliability becomes a property of the whole loop, from model behavior to organizational response.

A publication-grade mental model

A language-model system should be designed like a controlled stochastic process embedded in an organization. The model proposes continuations. The controller shapes probabilities. Sensors reveal drift. Policies allocate authority. Humans handle judgment. Metrics track accuracy, risk, burden, and recovery.

16. Try It Yourself

Small experiments make the bridge tangible.

Experiment 1: The probability tilt

- Ask a model to complete: “The bank was ...”
- Ask: “The fisherman sat by the bank. The bank was ...”
- Ask: “The accountant audited the bank. The bank was ...”

Lesson: The same word appears in different states. The probability distribution changes with the surrounding history.

Experiment 2: The reachability game

- Pick a target word, such as “zebra.”
- Try to make the next sentence end with that word using one control word.
- Repeat with three control words, then ten control words, then a full instruction.

Lesson: Prompt length and precision act like steering budget. Some targets become easier as budget grows.

Experiment 3: The feedback loop

- Ask for a small program.
- Run the tests or the code.
- Feed the error message back and ask for a repair.
- Repeat until the tests pass or the uncertainty requires a human decision.

Lesson: The model remains probabilistic. The loop becomes corrective.

Experiment 4: The entropy window

- Ask for ten product names with very low temperature, if your interface exposes sampling controls.
- Repeat with high temperature.
- Repeat with medium temperature and a clear constraint set.

Lesson: Low variation can be stale, high variation can drift, and bounded variation is often useful.

Experiment 5: The burden audit

- Choose an AI-assisted workflow.
- Count retries, human corrections, validator failures, tool errors, memory cleanup, latency, and unexplained decisions.
- Compare that burden with final task success.

Lesson: Final success can hide repair cost. Burden reveals the real control cost.

17. Field Guide and Mathematical Core

Formal terms become simple once the picture is clear.

Formal term	Primer name	Intuition
State x_0	Where the story is now	The context the model continues from.
Control input u	The nudge	Prompt, schema, retrieval, tool choice, memory edit, or review gate.
Output y	Where it goes next	A token, answer, action, tool call, or artifact.
Reference r	The planned route	The spec or desired trajectory the loop compares against.
Executor agent	Hands on the wheel	The brain (the model) plus the actuators that act on the world.
Control agent	The navigator	Sensors plus comparison that turn drift into orders.
Reachable set	The reachable map	Possible outputs under available controls and budget.
Controllability	Do the knobs matter?	The degree to which controls can steer desired outcomes.
Observability	Can the loop see?	Whether sensors reveal drift before high cost arrives.
Feedback	Look, compare, correct	Validators, tests, reviewers, monitors, and revised controls.
Entropy	Useful wiggle	Variation that enables search within boundaries.
Tail risk	The far-corner monster	Rare events with high cost.
Viable region	The safe channel	States and trajectories that remain useful, bounded, and observable.
Damping	Does the wobble shrink?	Whether errors fade through correction or amplify through the path.
Burden	The cleanup cost	Retries, human repair, maintenance, latency, and hidden work.
Lyapunov-style meter	The worry-meter	A quantity that should fall as the loop stabilizes.
Prompt registry	A shelf of rudders	Versioned controls for different states and risks.
Operating model	The control room	Roles, rituals, thresholds, ownership, and escalation.

The compact mathematical picture

A language model at generation time can be treated as a stochastic dynamical system. The state is the current context. The transition rule maps that state to a probability distribution over next tokens or actions. A control input modifies the state, the transition, or the allowed action set.

Compact equations

$$\text{state}[?] = \text{context}[?]$$

$$\text{output}[?]_{+1} \sim P(\text{output} \mid \text{state}[?], \text{control}[?])$$

$$\text{state}[?]_{+1} = \text{update}(\text{state}[?], \text{output}[?]_{+1}, \text{observation}[?]_{+1})$$

controller: choose control[?] from observations to keep trajectories in the viable region

For agents, replace “next token” with a richer next step: tool call, memory write, retrieval query, code execution, message, or final artifact. The mathematics becomes broader while the picture stays familiar: state, control, observation, uncertainty, cost, and feedback.

Design principles

- Name the state before choosing the control.
- Version prompts, schemas, evals, tool gates, and memory policies as controllers.
- Use reachable-set thinking: Define target, budget, knobs, and measurement.
- Place sensors before high-authority actions.
- Allocate entropy: wider for search, tighter for execution.
- Track burden as a first-class cost.
- Demand damping: Errors should shrink through the loop.
- Make tool transitions lawful, logged, reversible when possible, and gated by risk.
- Route ambiguous high-impact cases to human judgment.
- Review incidents as control failures and redesign the loop.

Final intuition

The bridge in one paragraph

A language model is a probability machine that evolves through tokens, actions, tool calls, and memory updates. The model is the brain at the helm: It chooses each move, and prompting is a control input because it changes which futures become likely. Feedback improves reliability by sensing drift and correcting the path against the reference. Agentic AI is harder because the path includes tools, memory, artifacts, humans, and changing context. The central engineering task is to build a closed-loop system that keeps the distribution of possible trajectories inside a viable region.

Ask three questions before deploying any language-model system:

- Which futures have we made likely?
- Which futures have we made rare?
- How quickly will we sense and correct drift?

18. Further Reading

Six doors out of the primer, from control theory to language-model agents.

- Karl Johan Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2nd edition, 2021. The standard gentle entry into sensing, feedback, and stability.
- Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific. The formal backbone for decision-making under uncertainty and cost.
- Ashish Vaswani et al. “Attention Is All You Need.” NeurIPS, 2017. The transformer architecture behind the box that weights continuations.
- Long Ouyang et al. “Training language models to follow instructions with human feedback.” NeurIPS, 2022. How feedback signals shape the probability field itself.
- Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models.” *ICLR*, 2023. Language-model agents as interleaved thought, action, and observation.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. Trajectories, reward, and correction, one level deeper.