Concept Containers as Representation-Level Regulation in Artificial Agents

AI / agent architectures

Tyson Jeffreys
Independent Researcher
tyson@staygolden.dev

Version 1.2 — February 11, 2026

Series note (2/3). This paper is Part 2 of a three-paper series on regime-level regulation in intelligent systems. Part 1 (Two-Regime Control) frames latent coordination vs. compensation in embodied control and introduces a baseline regulator layer. Here we generalize the framing to representation-level regulation via concept containers that amortize causal structure and reduce repeated recomputation. Part 3 translates the same principle into a product-facing objective for research systems: minimizing time-to-analysis.. Companion framing: *Regulatory Ground for Agentic AI* extends the same regulation view to runtime operating layers (budgets, uncertainty gating, rollback, and a global restraint signal). Concept containers fill the representation-level component of that stack.

Abstract

We introduce **concept containers**: reusable compressed representations that preserve causal structure while minimizing cognitive or computational cost. We argue that concept containers function as a form of **representation-level regulation** in intelligent agents: they reduce repeated recomputation, lower internal contention, and stabilize decision trajectories while maintaining (or improving) transfer. This framing unifies ideas from temporal abstraction, world modeling, retrieval, and energy-aware AI under a single design objective: **minimize time-to-usable-causal-structure per unit compute**.

We formalize concept containers as learned or constructed abstractions z that support intervention-relevant predictions while enabling compute savings. We propose measurable quality criteria (causal fidelity, transfer, and compute reduction at equal performance), outline architectural patterns for LLM agents and learned controllers, and offer falsifiable experiments. We also describe failure modes—especially over-compression—and practical guardrails (provenance, falsifiers, and confidence).

In the companion runtime-regulation framing (*Regulatory Ground for Agentic AI*), containers act as representation-level boundary conditions: they reduce internal thrash so exogenous budgets and uncertainty gating can operate with fewer interventions.

1. Introduction

Modern AI systems can generate fluent outputs and solve many tasks, yet they frequently exhibit:

- long deliberation chains and redundant reasoning
- high token/tool usage to reach stable conclusions
- frequent reversals ("thrash") under new evidence
- poor transfer when the surface form changes

A common interpretation is that agents need *more* reasoning. We propose the opposite diagnosis: agents often suffer from **insufficient compression of causal structure**. They repeatedly rebuild similar models rather than reusing stable abstractions.

This paper presents a simple thesis:

> **Efficient intelligence requires representations that are both compressive and intervention-valid.**

We call these representations **concept containers**, and we treat them as a regulatory primitive—analogous to a baseline regulator that prevents chronic over-activation.

2. Concept containers

2.1 Definition

A **concept container** is:

> A reusable representation that preserves causal structure while minimizing computational cost.

It is not merely a summary, label, or topic tag. A container must support *correct action selection* under relevant interventions.

Formally, let x denote observations (text, sensory streams, states), a denote an intervention/action, and y denote outcomes of interest (success, safety, reward, constraint satisfaction). A container is a mapping f:

$$z = f(x_{1:t})$$

such that for a target intervention set $\mathcal{A}$, the representation z is sufficient to predict the effect of interventions:

$$\forall a \in \mathcal{A}: \quad p(y \mid do(a), x_{1:t}) \approx p(y \mid do(a), z)$$

This captures "preserves causal structure" as intervention-relevant sufficiency.

2.2 Properties

A good container has three operational properties:

1) **Compression**

- reduces effective dimensionality / state complexity
- lowers the cost of inference, planning, or deliberation

1) **Portability**

- generalizes across surface forms and contexts
- supports transfer without re-deriving the full model

1) **Procedural force**

- changes downstream behavior (policy choice, search strategy, uncertainty handling)
- acts like a mental macro or planning primitive

2.3 Container vs summary

A summary compresses content. A container compresses **structure**.

- Summary answers: "What was said?"
- Container answers: "What variables matter? How do they interact? What levers change outcomes? What would falsify this?"

3. Containers as representation-level regulation

3.1 Repeated recomputation and internal contention

Consider an agent producing a sequence of internal states h_t while solving a task. Define a proxy for internal activation/effort $x(t)$ (tokens generated, tool calls, planner expansions, or compute time).

Many agents repeatedly reconstruct similar intermediate models:

- the same causal argument rederived across prompts
- redundant search and rechecking
- repeated "stabilization" steps after perturbations

This is **internal contention**: the system uses capacity to manage instability rather than to improve representation quality.

3.2 Containers reduce activation duty cycle

Let $\bar{x}(t)$ be a filtered internal-load signal. Define a high-activation indicator:

$$H(t) = \mathbb{1}[\bar{x}(t) > \theta]$$

and an activation duty cycle:

$$DC(t_0, t_1) = \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} H(t)\, dt$$

Hypothesis: If an agent uses good containers, it achieves comparable task success while reducing DC and peak $x(t)$ by reusing causal structure rather than recomputing it.

3.3 Regulation framing

We interpret containers as a regulator at the representation layer:

- they lower baseline compute required to stay coherent on a problem
- they reduce thrash and reversals
- they enable "spikes" of compute only when uncertainty truly demands it

3.4 Interface to runtime regulation layers

Representation-level regulation and runtime regulation solve complementary problems. A runtime regulator (budgets, uncertainty gating, rollback) can keep an agent inside safe operating bands, but it does not by itself reduce repeated recomputation. Containers provide reusable causal structure that lowers the internal-load signal the regulator would otherwise be reacting to.

Two practical integration points:

- **Posture-modulated reuse:** a global restraint/posture variable can modulate retrieval and recomputation thresholds. Under tighter posture, prefer high-provenance containers and limit novel exploration; under looser posture, allow synthesis of new containers and broader search.
- **Regulated container writes (commit + abstention):** treat container creation/update as a *committed action* (a write to the agent's long-lived causal cache), and judge the write under an *abstention* policy rather than "always pick one."
  - **Critic as ranker, not decider:** use a judge/critic to *rank candidate containers* and estimate uncertainty, not to force a single winner or author the update.
  - **High-tie / abstain ⇒ no write:** if candidates are near-tied, the critic is uncertain, or falsifiers are missing, *do not* write a new container. Instead, request evidence, generate discriminating falsifiers, or run a minimal test first.
  - **Collapse prevention:** this links "critic collapse / oscillation" (forced selection under uncertainty) to container-level failure modes (over-compression and thrash). Abstention turns uncertainty into an evidence-seeking step rather than a brittle commit.

  Require provenance, explicit falsifiers, and confidence; log revisions and measure whether updates persist and constrain future trajectories.

4. Compute and energy implications

Large-scale systems pay a direct cost for repeated recomputation: tokens, tool calls, and model forward passes translate into compute time and energy.

Let k be a compute proxy (tokens, tool invocations, FLOPs). Total compute over a task:

$$K = \int_0^T k(t)\, dt$$

If containers reduce long tails of deliberation and repeated stabilization, then K decreases at equal task success.

A second-order benefit is variance reduction: fewer thrash cycles means more stable trajectories in internal state space, which reduces the need for corrective computation.

5. Quality criteria and metrics

We propose three core criteria for container quality.

5.1 Causal fidelity

A container must preserve intervention-relevant structure. Practical tests:

- counterfactual queries: does the container predict changes under plausible interventions?
- consistency checks: do predictions remain stable across paraphrases?
- falsifier sensitivity: does new disconfirming evidence correctly update the container?

5.2 Transfer

Test whether containers generalize across:

- domains (technical → business → safety)
- surface forms (different wording, different evidence ordering)
- task variants (similar causal skeleton)

5.3 Compute reduction at equal performance

For a fixed success threshold, measure:

- tokens / tool calls / planner expansions
- time-to-stable-decision (time-to-analysis)
- reversal count (how often conclusions flip)

A container that increases compute while improving success may still be valuable, but it is not functioning as a regulation primitive.

5.4 Failure mode: over-compression

Over-compression collapses the causal skeleton:

- the container becomes a slogan, label, or vibe
- it stops making intervention-valid predictions
- it becomes brittle and misapplies across contexts

A useful diagnostic: if a container cannot produce falsifiers, it is likely over-compressed.

5.5 Longitudinal revision persistence

To align with the "endogenous revision" frontier for discovery-capable agents, evaluate whether container updates *persist* and constrain future behavior (not just patch the last answer). Practical metrics:

- **Revision half-life:** how long (or how many episodes) a revision remains active before being overwritten or ignored.
- **Reversal count:** how often the container flips back and forth under similar evidence.
- **Constraint imprint:** whether the revised container measurably changes downstream planning (tool choices, search breadth, or abstention) in later tasks.

6. Architectural patterns

6.1 Container bank architecture

A practical agent architecture:

- **Detector:** identifies when a task is entering repeated recomputation (thrash signals)
- **Synthesizer:** constructs a container z from evidence and reasoning traces
- **Bank:** stores containers with provenance and scope
- **Retriever:** fetches candidate containers for new tasks
- **Composer:** adapts/merges containers and generates action guidance

The key shift is that the system stores **causal abstractions**, not just answers.

6.2 A minimal container schema

A container should include:

- **Name / handle**
- **Scope** (where it applies)
- **Causal skeleton** (variables + directional relations)
- **Levers** (interventions that matter)
- **Predictions** (what changes under each lever)
- **Falsifiers** (what evidence would break it)
- **Confidence** and **provenance** (sources, timestamps, assumptions)
- **Cost signature** (what compute it saves; what it may risk)

6.3 Integration with LLM agents

For LLM-based agents, containers can be implemented as structured artifacts produced after an expensive reasoning episode and re-used via retrieval:

- After a task: produce a container instead of a long transcript.
- On new tasks: retrieve containers and start from them.
- Penalize re-derivation: add a training or runtime objective that prefers reuse when valid.

6.4 Integration with planners and controllers

In classical or learned control, containers correspond to:

- options / skills (macro-actions)
- state abstractions (task-relevant variables)
- invariants (constraints that govern safe/efficient behavior)

The container view emphasizes causal sufficiency and reuse as an efficiency and stability primitive.

6.5 Regulated agent stack integration

In a regulated agent stack (untrusted competence core + independent runtime regulator), containers can be treated as first-class objects with explicit cost/risk semantics:

- **Cost signature as input to budgets:** container metadata (expected compute savings, expected failure modes) can inform budgeting and gating decisions.
- **Containers as boundary conditions:** invariants and "must-hold" causal commitments can be stored in containers and enforced by the runtime regulator during tool use or actuation.
- **Auditability:** container retrieval, composition, and revision events should be logged so governance can reason about when a representational change caused a behavioral change.

7. Falsifiable experiments

7.1 Experiment A: research-to-decision tasks

Construct a benchmark where an agent must form a stable causal model from multiple sources.

Compare:

- A1: baseline agent (retrieve + answer)
- A2: summarize sources
- A3: container pipeline (extract causal skeleton + levers + falsifiers)

Measure:

- success / correctness
- tokens + tool calls
- reversal count
- time-to-stable-decision

Prediction: A3 reduces compute and reversals at equal success.

7.2 Experiment B: paraphrase and domain transfer

Hold the causal structure constant while varying surface form.

Measure whether the container pipeline maintains:

- stable levers and predictions
- stable falsifiers
- lower recomputation costs

7.3 Experiment C: induced perturbation / conflicting evidence

Inject adversarial or conflicting evidence mid-task.

Measure:

- whether the agent updates the container correctly
- whether the revision persists and constrains later exploration (revision half-life)
- whether it thrashes (multiple reversals)
- compute used to regain coherence

Negative test: If containers cause higher brittleness under conflict, the system requires stronger provenance and falsifier mechanisms.

8. Discussion

Concept containers are not a replacement for reasoning; they are a method for **storing the results of reasoning as reusable causal structure**.

The core claim is architectural:

- Many agents waste compute not because they cannot think, but because they cannot **retain and reuse the right kind of thinking**.
- Containers reduce the need for chronic corrective computation.
- Over time, a container bank can act like a representational baseline: stable, low-cost, and ready to support spikes when needed.

Seen through the runtime-regulation lens, concept containers reduce the probability that an agent will enter high-activation regimes that trigger frequent intervention (budgets, safe-mode switches, rollback). In other words: better representations make band-limited optimization easier to enforce, because the regulator is not constantly compensating for internal instability.

9. Limitations

- Defining $\mathcal{A}$ (the relevant intervention set) is domain-dependent.
- Containers can drift if not grounded in provenance.
- Poor retrieval can misapply containers.
- Over-compression is a persistent risk.

10. Conclusion

We proposed concept containers as a representation-level regulation mechanism for artificial agents. A concept container is a reusable compressed representation that preserves causal structure while minimizing computational cost. This framing yields concrete design principles, measurable quality criteria, and falsifiable predictions.

The broader implication is that efficient intelligence depends not only on the ability to reason, but on the ability to **stabilize and reuse causal structure**—reducing repeated recomputation and internal contention while preserving intervention-valid competence.

Appendix A: Practical scoring functions

A pragmatic container score can combine:

$$Q(z) = \alpha\,\mathrm{Fidelity}(z) + \beta\,\mathrm{Transfer}(z) - \gamma\,\mathrm{Compute}(z)$$

where Compute(z) is measured as additional tokens/tool calls required to solve a fixed task set with vs. without container reuse.

Appendix B: Example container (template)

- **Handle:** "Pressure point"
- **Scope:** systems strategy / pipeline optimization
- **Causal skeleton:** upstream bottlenecks create downstream compounding constraints
- **Levers:** reduce bottleneck latency; increase reuse of analysis structure
- **Predictions:** lowering time-to-analysis improves outcomes across decision layers
- **Falsifiers:** if time-to-analysis decreases but decisions do not improve, the bottleneck is elsewhere
- **Provenance:** sources + assumptions

Example container (instantiated):

- **Handle:** "Latent coordination vs. compensation"
- **Scope:** embodied control stacks (locomotion/manipulation)
- **Causal skeleton:** when passive dynamics + low-bandwidth feedback can carry the task, corrective effort stays low; when coupling breaks (uncertainty/contact/limits), overrides dominate and energy rises.
- **Levers:** measure compensation load; add a slow regulator that biases posture/planning/ estimation back toward low-compensation basins.
- **Predictions:** lowering compensation duty cycle reduces energy per task and peak actuator stress at similar success.
- **Falsifiers:** if duty cycle falls without reducing energy/stress or improving robustness, the index is mis-specified or the bottleneck lies elsewhere.
- **Provenance:** system logs (torque/current, replanning events, estimator uncertainty, slip), plus task/terrain conditions.

Note on authorship and tools:
This work was developed through iterative reasoning, modeling, and synthesis. Large language models were used as a collaborative tool to assist with drafting, clarification, and cross-domain translation. All conceptual framing, structure, and final judgments remain the responsibility of the author.