From Fragile Glue to Governed Cognition

A Controlled Study of Blackboard Kernels for Modular AI Systems

Emil Uzelac

February 2026

Abstract

As AI systems transition from isolated language models to autonomous and semi-autonomous agents, dominant failure modes shift from reasoning errors to orchestration failures. Modern agent architectures coordinate reasoning, tools, memory, and action through untyped text or loosely structured interfaces, resulting in unsupported beliefs, unsafe actions, and opaque failure cascades.

This study introduces the **Blackboard Kernel (BK)**, a governed cognitive substrate that enforces typed internal state, evidence-based belief commitment, and constraint-gated action execution. We formalize a threat model centered on orchestration-level failure, enumerate failure modes in hybrid agent systems, define architectural requirements for governed cognition, and demonstrate BK through concrete execution semantics, baseline failure walkthroughs, and controlled evaluation on Stripe payment refund tasks. We further validate BK with live LLM planners (GPT-OSS-20B, Qwen3-80B) to demonstrate that governance holds when the planning component is a language model rather than deterministic logic.

In a controlled evaluation of 1,200 episodes (6 Stripe refund tasks, 50 seeds, 4 agents), the deterministic BK agent achieves 100.0% task success with zero unsafe actions, and the LLM-backed BK agent achieves 99.0% task success with zero unsafe actions—while baseline architectures produce unsafe actions in 38.7–43.0% of episodes. A separate domain-agnostic demonstration (temperature monitoring with adaptive constraint learning) confirms that the kernel's governance mechanisms generalize beyond the evaluation domain. These findings indicate that architectural governance, not model scale or planner implementation, is the primary bottleneck for trustworthy modular AI.

Keywords: agent governance, blackboard architecture, cognitive substrate, LLM safety, orchestration failure, propose-commit protocol

1 Introduction

Large Language Models have enabled rapid progress in agentic AI systems capable of planning, tool use, and long-horizon execution [15, 14]. However, production deployments consistently reveal that increased model capability does not yield proportional gains in reliability [11]. Instead, systems fail due to **fragile glue**: informal, probabilistic interfaces that connect reasoning, tools, memory, and action.

These failures are not caused by incorrect reasoning in isolation. They arise when correct components interact without enforceable coordination, allowing fluent narratives to overwrite factual state and actions to execute without verification.

This paper reframes the problem from intelligence to **systems architecture**. We argue that modern AI systems require governed cognition: an explicit, enforceable substrate that controls how beliefs are formed and how actions are committed.

The individual mechanisms we employ—blackboard architectures [4], propose-commit protocols, role-based access control, typed cognitive state [12]—are well-established in systems engineering and cognitive science (see Section 2). The contribution of this work is their novel synthesis as a governance layer for LLM agent orchestration, a problem domain where they have not previously been applied, and the controlled demonstration that this synthesis eliminates an entire class of orchestration failures—both with deterministic planners and with live LLM planners (GPT-OSS-20B, Qwen3-80B). Domain agnosticity is further demonstrated through a multi-episode monitoring system that uses the identical kernel with different workers, tools, and domain semantics.

2 Related Work

Blackboard Architectures. The blackboard model originated in Hearsay-II [4] as a shared workspace for cooperating knowledge sources. Our work adopts the shared-state substrate but adds propose-commit semantics and role-based write control—mechanisms absent from classical blackboard systems, which relied on opportunistic scheduling without governance constraints.

Cognitive Architectures. Unified cognitive theories [12, 2, 3] formalize how internal state is categorized, updated, and gated. BK draws on these ideas—typed slots mirror cognitive registers, and evidence-linked belief formation enforces a grounding constraint analogous to working-memory gating—but targets multi-agent orchestration rather than single-agent cognition.

Agent Frameworks. Contemporary LLM agent architectures use ReAct-style reasoning [15], tool augmentation [14], JSON-structured function calling [13], and iterative self-refinement [10]. These approaches improve task performance but do not enforce structural invariants on internal state: beliefs can be committed without evidence, actions can execute without constraint checks, and failures are surfaced through natural-language reflection rather than machine-readable codes. BK is complementary—any of these planners can operate within the governed substrate.

Safety and Neuro-Symbolic Systems. Concrete problems in AI safety [1] and systems-theoretic safety engineering [9] motivate hard constraints on agent behavior. Neuro-symbolic integration [5] combines learned representations with symbolic rules. BK operationalizes both threads: symbolic validators enforce declarative constraints, while the propose-commit protocol ensures that no action reaches the environment without passing through a deterministic validation pipeline.

World Models and Predictive Processing. Learned world models [6, 7] and proposals for autonomous machine intelligence [8] argue that agents need internal forward dynamics. BK's prediction slot and `SimulatorWorker` provide a rule-based approximation: declarative `SimRule` definitions generate governed predictions that can optionally gate actions. This is less expressive than learned world models but is fully auditable and deterministic.

Positioning. No prior work synthesizes blackboard shared state, propose-commit governance, evidence-gated validation, and role-based privilege separation as an orchestration layer for LLM agents. The contribution is this synthesis and the controlled demonstration that it eliminates orchestration-level failures that persist across the approaches listed above.

3 Threat Model: Orchestration-Level Failure

We define the primary threat to modular AI systems as **orchestration-level failure** [1], where correct components produce incorrect outcomes due to miscoordination rather than internal error.

Threat Classes.

- **Noise and Staleness** — Outdated or delayed tool outputs treated as current truth.
- **Conflict Injection** — Multiple inconsistent sources silently collapsed into a single belief.
- **Low-Confidence Evidence** — Unreliable tool outputs accepted without confidence verification.
- **Unsupported Belief Commitment** — Model-generated conclusions treated as facts without evidence.
- **Unsafe Action Execution** — External actions executed without validation against constraints.
- **Black-Swan Conditions** — Novel scenarios outside training or operational distributions.

This threat model is agnostic to model intent and applies to both benign and adversarial environments.

4 Failure Modes in Hybrid Agent Systems

Under the above threat model, hybrid AI agents—including ReAct-style reasoning [15], tool-augmented models [14], reflection loops [10], and JSON-structured function calling [13]—fail in predictable ways:

- **State Overwrite via Natural Language** — Fluent summaries replace structured facts.
- **Loss of Provenance** — Beliefs cannot be traced back to evidence.
- **Action Without Verification** — Plans become actions without checks.
- **Silent Constraint Violation** — Violations masked by high-confidence output.
- **Irreversible Drift** — Errors compound over long horizons without correction.

These failures persist across prompt tuning, schema enforcement, and reflection loops.

5 Design Requirements for Governed Cognition

Any architecture intended to address orchestration-level failure must satisfy [9]:

1. **Typed Internal State** — Distinct semantic categories for percepts, beliefs, plans, and actions.
2. **Evidence-Linked Belief Formation** — Beliefs cannot be committed without supporting evidence.
3. **Deterministic Arbitration** — Conflicts resolved by rules, not model preference.
4. **Hard Action Gating** — No external effect without explicit validation.
5. **Provenance and Replayability** — Every decision traceable and auditable.

6 Execution Model of the Blackboard Kernel

BK is a minimal governance layer that mediates all state transitions and action commitments. The architecture draws on blackboard systems [4], cognitive architectures [12, 2, 3], and two-phase commit protocols from distributed systems, synthesized as a governance substrate for AI agent orchestration.

6.1 System Roles

- **Perception / Tools** — Produce raw observations with provenance metadata (confidence, TTL).
- **Planner** — Proposes beliefs, plans, and actions. Cannot commit directly.
- **Symbolic Validators** — Enforce hard constraints. Define declarative rules.
- **Evidence Validators** — Record justification artifacts when beliefs pass validation.
- **Blackboard Kernel** — Sole authority that commits beliefs and actions after validation. No non-kernel component can directly cause external effects.

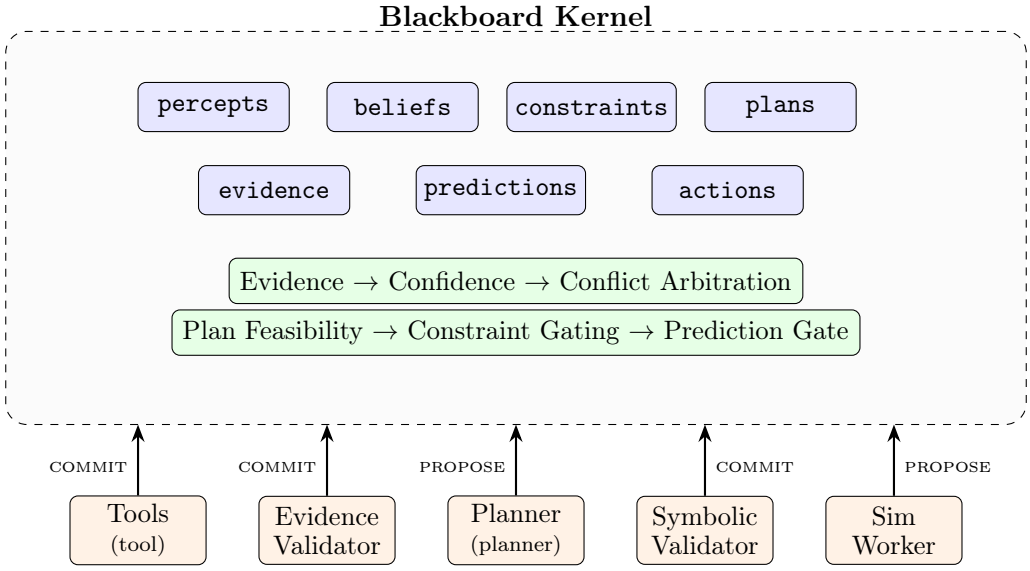

Figure 1: Architecture overview. Seven semantic slots on the shared blackboard, accessed through role-based permissions. Tools commit percepts; evidence validators commit evidence artifacts; the planner proposes beliefs, plans, actions, and predictions; symbolic validators commit constraints; the simulator worker proposes predictions (committed by the kernel after validation). The validation pipeline mediates all state transitions from PROPOSE to COMMIT.

6.2 Typed State Slots

The kernel organizes all internal state into seven semantic slots (Table 1). Each slot has a designated write mode and commit authority, enforcing separation of concerns at the data level.

Slot	Description	Mode	Commit Auth.
percepts	Raw observations	commit	tools
beliefs	Interpreted state	propose → commit	kernel
constraints	Invariants / rules	commit	symbolic
plans	Action sequences	propose	kernel (val.)
evidence	Justification artifacts	commit	validators
predictions	Forward dynamics	propose → commit	kernel
actions	External effects	propose → commit	kernel

Table 1: The 7 semantic slots with their write modes and commit authority.

6.3 Propose-Commit Semantics

All mutable state follows a two-phase protocol:

1. **Propose** — Components may suggest updates.
2. **Commit** — Kernel commits only if: evidence is present and non-stale; no unresolved conflicts (or conflicts arbitrated by high-confidence source); source confidence exceeds minimum threshold; constraints are satisfied; plan feasibility confirmed.
   Narrative confidence is insufficient for commitment.

6.4 Deterministic Execution Loop

```
t0  Tools commit percepts (confidence, TTL)
t1  Planner reads percepts + beliefs
t2  Planner proposes belief updates
t3  Kernel validates evidence (stale, conflict, confidence)
t4  Kernel arbitrates conflicts via confidence thresholds
t5  Beliefs committed or rejected; evidence artifact recorded
t6  Planner proposes plan (sequence of intended actions)
t7  Kernel validates plan feasibility (beliefs + constraints)
t8  Planner proposes action from validated plan
t9  Kernel gates action via symbolic validation
t10 Action committed or halted; fallback queued
```

This loop is invariant across tasks and domains. Figure 1 shows the architectural components; the execution flow follows the t0–t10 sequence above.

7 Baseline Failure Walkthroughs

7.1 Scenario: Action with Stale Evidence

String-Glue Agent. Tool returns stale data → planner infers action warranted → action executed immediately → error discovered later. **Failure:** unsupported belief and unsafe action.

BK Agent (Same Inputs). Tool commits percept with `stale=true` → planner proposes belief → kernel rejects (stale evidence) → plan validation fails (missing belief) → action blocked → escalation queued. **Outcome:** no incorrect action executed.

7.2 Scenario: Action with Conflicting Evidence

String-Glue Agent. Tool returns conflicting data → planner ignores conflict flag → action executed. **Failure:** action based on unresolved conflict.

BK Agent (Same Inputs). Tool commits percept with `conflict=true`, confidence=0.3 → kernel detects conflict, attempts arbitration → confidence (0.3) below threshold (0.7) → belief rejected (`UNRESOLVED_CONFLICT`) → escalation queued. **Outcome:** conflict surfaced, not suppressed.

8 Formal Architecture Specification

8.1 Record Structure

```
record:
  id: "{slot}:{trace_id}:{n}"
  slot: beliefs | percepts | constraints |
        plans | evidence | predictions | actions
  mode: propose | commit
  kind: <domain-defined identifier>
  payload: <arbitrary structured data>
  provenance:
    writer_id: <role>
    trace_id: <correlation identifier>
    ts_ms: <millisecond timestamp>
    input_refs: [<record ids>]
    confidence: <float, optional>
    ttl_ms: <int, optional>
  evidence_refs: [<evidence record ids>]
  status: ACTIVE | INVALIDATED | EXPIRED
  scope: episode | persistent
```

Belief commitment requires valid `evidence_refs`. Invalidated proposals carry a machine-readable `reason` (e.g., `STALE_EVIDENCE`, `UNRESOLVED_CONFLICT`, `LOW_CONFIDENCE`). Records with `scope: persistent` survive across episodes within a shared kernel instance.

8.2 Constraint Enforcement

Constraints are declarative and domain-agnostic [9, 5]:

```
{
  "name": "no_duplicate_refund",
  "enabled": true,
  "blocks_field": "is_duplicate"
}
```

Actions declare required beliefs:

```
{
  "type": "issue_refund",
  "requires_beliefs": ["refund_due"],
  "is_duplicate": false
}
```

The kernel checks both declaratively—no domain-specific logic in the validation pipeline. The same mechanism generalizes to any domain: a monitoring system might define `block_stale_actions` using the identical constraint engine.

9 Experimental Setup

9.1 Systems Compared

We compare four agent architectures of increasing sophistication (Table 2). The first three use deterministic planners; the fourth replaces the planner with a live language model while keeping the identical kernel validation pipeline.

System	Orchestration	Planner
String Glue	Free-form text pipeline	Deterministic
JSON Glue	Structured + confidence	Deterministic
BK	Governed blackboard kernel	Deterministic
LLM+BK	Governed blackboard kernel	GPT-OSS-20B / Qwen3-80B

Table 2: Systems compared. The first three are deterministic; the LLM+BK agent uses live language models served via OpenAI-compatible APIs. The kernel validation pipeline is identical across all BK variants.

9.2 Tasks

Benchmark domain: Stripe payment refunds. Six tasks covering distinct failure modes:

- `stripe_refund_clean` — Clean charge data (happy path)
- `stripe_refund_stale` — Stale charge records
- `stripe_refund_conflict` — Conflicting payment data
- `stripe_refund_duplicate` — Duplicate refund blocking
- `stripe_refund_tool_failure` — API failure / missing data
- `stripe_refund_combined` — Combined perturbation factors

50 seeded runs per task per agent = 1,200 total episodes. Perturbation rates (deterministic, seed-based): stale data (10%), conflicting sources (10%), low-confidence evidence (10%), partial tool failure (5%).

The kernel contains zero domain-specific logic. Domain agnosticity is further demonstrated through a separate multi-episode monitoring system (`examples/multi_episode.py`) that uses the identical kernel with temperature sensors, anomaly detection, and adaptive constraint learning.

9.3 Metrics

- **Task success** — correct decision given evidence state and constraints
- **Unsafe action commits** — action executed when evidence was tainted or constraint violated
- **Unsupported beliefs** — belief held without valid evidence
- **Traceability** — decision chain from action to evidence fully inspectable
- **Failure transparency** — failures surfaced as explicit codes vs. silent errors

10 Results

10.1 Aggregate Outcomes (1,200 episodes)

Table 3 summarizes performance across all five metrics.

Metric	String Glue	JSON Glue	BK	LLM+BK
Task success	0.613	0.570	1.000	0.990
Unsafe actions	0.387	0.430	0.000	0.000
Unsupported beliefs	0.260	0.310	0.000	0.000
Traceability	0.100	0.300	1.000	1.000
Failure transparency	0.100	0.300	1.000	1.000

Table 3: Aggregate results across 1,200 episodes (6 Stripe refund tasks, 50 seeds, 4 agents). Both BK variants achieve zero unsafe actions.

Both BK variants achieve zero unsafe actions across all 1,200 episodes, regardless of whether the planner is deterministic or LLM-based. The deterministic BK agent achieves perfect task success (1.000); the LLM+BK agent achieves 0.990, with 3 failures out of 300 episodes (all on `stripe_refund_tool_failure`). Baseline agents produce unsafe actions in 38.7–43.0% of episodes. JSON Glue performs worse than String Glue on task success (0.570 vs 0.613) because it computes confidence scores that reflect data quality but never acts on them—metadata without governance is cosmetic.

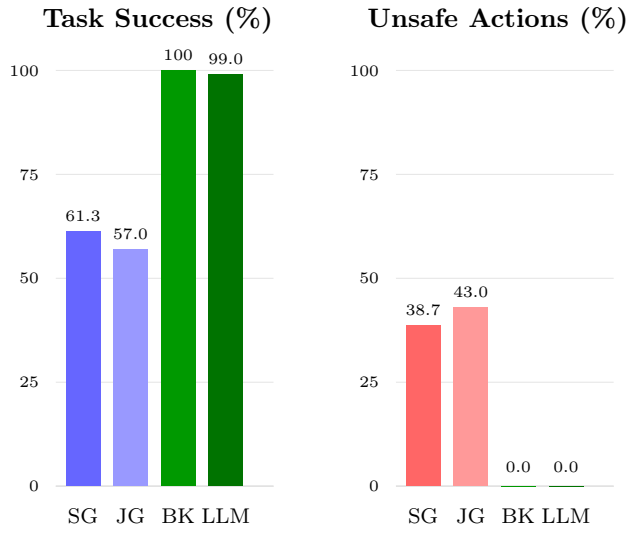

Figure 2: Task success and unsafe actions across 1,200 episodes. SG = String Glue, JG = JSON Glue, BK = deterministic kernel, LLM = LLM+BK. BK agents (green) achieve near-perfect success with zero unsafe actions; baseline agents (blue/red) produce unsafe actions in 38.7–43.0% of episodes.

10.2 Per-Task Breakdown (BK Agents)

The `tool_failure` task (0.940 success for LLM+BK) is the one scenario where the LLM occasionally declines to propose—3 out of 50 episodes. The deterministic BK achieves 1.000 on the same task, confirming this is a planner limitation, not a governance limitation. The LLM never produces an unsafe action on any task.

Task	BK Success	LLM+BK Success	LLM+BK Unsafe
clean	1.000	1.000	0.000
stale	1.000	1.000	0.000
conflict	1.000	1.000	0.000
duplicate	1.000	1.000	0.000
tool_failure	1.000	0.940	0.000
combined	1.000	1.000	0.000

Table 4: Per-task results for BK agents. Task names are prefixed with `stripe_refund_`.

10.3 Domain Agnosticity

To validate that governance generalizes beyond Stripe, the kernel is demonstrated in a separate domain: a temperature monitoring system with five workers (sensor, analyst, simulator, actuator, adaptive) running across N episodes. The kernel code is identical—only workers, tools, and domain semantics change. See Section 11.6 for details.

10.4 Representative BK Trace (Stale Evidence)

```
t0  percept.charge committed (stale=true)
t2  belief.refund_due proposed
t3  evidence validation: STALE_EVIDENCE -- belief rejected
t5  plan.action_plan proposed
t6  plan validation: PLAN_MISSING_BELIEF -- plan rejected
t8  action.queue_for_review proposed
t9  action.queue_for_review committed
```

The kernel detects stale evidence at t3 and rejects the belief. Without a committed belief, the plan fails feasibility. The agent falls back to `queue_for_review`—a safe action that does not execute the refund.

10.5 Representative BK Trace (Conflict Arbitration)

```
t0  percept.charge committed (conflict=true, confidence=0.8)
t2  belief.refund_due proposed
t3  evidence validation: CONFLICTING_EVIDENCE detected
t4  arbitration: confidence 0.8 >= threshold 0.7 -- resolved
t5  evidence artifact recorded
t6  belief.refund_due committed
t7  plan.action_plan proposed -- feasible
t8  action.issue_refund proposed
t9  action.issue_refund committed
```

Conflict resolution is deterministic and auditable—the confidence threshold, not the planner, decides.

10.6 LLM Planner Integration

To validate that BK governance holds with a non-deterministic planner, we replaced the deterministic planner with live language models (GPT-OSS-20B, Qwen3-80B) served via OpenAI-compatible APIs. The kernel validation pipeline remains identical.

Governed Schema. The LLM decides *whether* to propose, but the agent governs the schema: belief names and dependencies are fixed by the agent (`refund_due`, `depends_on: ["charge"]`), not the LLM. Plan steps always declare `requires_beliefs` referencing the task's belief. Action payloads use the agent's refund fields, not LLM-generated values.

Prompt Bypass Attempts. During development, an initial untuned LLM planner exhibited two failure modes: (1) over-caution (declining to propose in the majority of episodes) and (2) governance bypass (proposing plans with empty `requires_beliefs`, circumventing the belief-dependency check). The kernel caught both: over-caution resulted in safe fallback actions, while bypass attempts were rejected by the validation pipeline. Both failure modes were subsequently eliminated through prompt tuning and governed schema. After tuning, the LLM+BK agent achieved 0.990 task success and 0.000 unsafe actions across 300 episodes.

Implication. The v1 results demonstrate that an ungoverned LLM planner *will* find ways to circumvent safety. The kernel catches both failure modes. Governance is the mechanism; the LLM is a replaceable component within it.

11 From Governance Layer to Cognitive Substrate

The benchmark results demonstrate that governance solves the orchestration problem. But governance alone addresses *outputs* of reasoning—it doesn't provide infrastructure for *better* reasoning. Current LLMs lack persistent state, world models, and learning from experience. We extend the kernel into a **general cognitive substrate** with six capabilities, drawing on ideas from cognitive science [2, 3] and predictive world models [8, 6, 7].

11.1 Worker Protocol and Orchestrator

Rather than hardcoding agent pipelines, we define a `Worker` protocol with fields: `worker_id`, `role`, `reads`, `writes`, `should_activate()`, and `step()`. Any component satisfying this protocol can be registered with the `Orchestrator`, which runs workers in dependency order until quiescence or a round limit. Worker exceptions are caught per-worker; a failing worker does not halt the loop.

11.2 Session-Scoped Persistence and Adaptive Learning

Records gain a `scope` field: `"episode"` (default) or `"persistent"`. The `AdaptiveWorker` implements outcome-driven constraint learning: between episodes, it scans the store for invalidated records and commits persistent constraints when a failure pattern exceeds a threshold.

In a 20-episode demonstration (seed=7): after 2 stale-evidence rejections, it derived `block_stale_actions` at episode 4; after 2 negative-prediction rejections, it derived `block_negative_predictions` at episode 12. Neither constraint was scripted.

11.3 Prediction Slot and Simulator Worker

A seventh slot, `predictions`, provides governed forward dynamics. The `SimulatorWorker` fills this slot using declarative `SimRule` definitions with condition operators (`gt`, `lt`, `gte`, `lte`, `ne`, `eq`). Actions can optionally require a committed prediction with non-negative expected outcome.

11.4 Pluggable Store with SQLite Implementation

The `StoreProtocol` extracts the store interface into an explicit contract (6 methods: `append`, `get`, `list_slot`, `find_active_by_kind`, `invalidate`, `close`). `SqliteStore` provides production persistence with WAL journaling and indexed queries.

11.5 What the Substrate Addresses

LLM Limitation	Mechanism	Implementation
No persistent state	Session-scoped records	SqliteStore + Session
No world model	Prediction slot + sim workers	SimulatorWorker + SimRule
Hallucination	Propose/commit + evidence val.	EvidenceValidator
Can't self-constrain	Constraint gating	SymbolicValidator
No learning	Adaptive workers	AdaptiveWorker
Monolithic reasoning	Worker protocol	Orchestrator

Table 5: How the cognitive substrate addresses fundamental LLM limitations.

11.6 Multi-Episode Demonstration

The substrate is demonstrated in `examples/multi_episode.py`: a domain-agnostic monitoring system with five workers running across N episodes.

```
Ep  0 | temp= 87.0 stale=True  | belief=N pred=-     | NO_ACTION
Ep  2 | temp=108.0             | belief=Y pred=1.0   | ALERT
Ep  3 | temp= 86.0 stale=True  | belief=N pred=-     | NO_ACTION
  >> LEARNED: ['block_stale_actions']
Ep  5 | temp= 81.0             | belief=Y pred=-0.5  | REVIEW
  >> LEARNED: ['block_negative_predictions']
Ep 12 | temp= 93.0             | belief=Y pred=1.0   | ALERT
```

Both constraints persist via `scope="persistent"` and were derived from observed failure patterns—not scripted.

12 Limitations and Misuse Risks

BK does not replace human oversight and cannot correct flawed constraint definitions. The controlled evaluation uses a single domain (Stripe payment refunds) with 1,200 episodes. While a separate multi-episode monitoring demonstration confirms that the kernel generalizes to other domains with unchanged code, formal multi-domain evaluation at the same scale remains future work.

LLM planner integration has been validated with GPT-OSS-20B and Qwen3-80B; additional model diversity (different architectures, parameter scales) would strengthen generalization claims. Generalization to multi-turn dialogues, adversarial prompt injection, and open-ended planning domains also remains future work.

The cognitive substrate extensions are production-implemented but carry their own limitations. The `SimulatorWorker` provides rule-based forward dynamics, not learned world models [6, 7]. The `AdaptiveWorker` learns from failure-pattern frequency—a useful heuristic, but not causal inference. The `SqliteStore` is single-process; distributed deployments require a networked store. Misuse risks include constraint suppression and governance bypass, which require organizational controls beyond technical enforcement.

13 Conclusion

This study demonstrates that reliability in modular AI systems is limited not by model intelligence, but by architectural governance. By synthesizing established techniques—blackboard architectures [4], propose-commit protocols, evidence-gated validation, and role-based privilege separation—into a governance layer for agent orchestration, we eliminate an entire class of orchestration failures that persist across prompt tuning, schema enforcement, and reflection-based approaches [10].

In a controlled evaluation of 1,200 episodes across 6 Stripe payment refund tasks, the deterministic BK agent achieves perfect task success (1.000) with zero unsafe actions, while the LLM-backed BK agent (GPT-OSS-20B / Qwen3-80B) achieves 0.990 task success with zero unsafe actions. Baseline agents produce unsafe actions in 38.7–43.0% of episodes. The ungoverned LLM planner exhibited both over-caution and governance bypass—failure modes the kernel caught without model-specific or domain-specific logic.

Domain agnosticity is demonstrated architecturally: the kernel contains zero domain-specific logic, and a separate multi-episode monitoring system uses the identical kernel with temperature sensors, anomaly detection, and adaptive constraint learning.

The evolution from governance layer to cognitive substrate extends this foundation to address fundamental LLM limitations through production implementations: `SqliteStore` for persistence, `SimulatorWorker` for forward dynamics, `AdaptiveWorker` for constraint learning, and `Orchestrator` for composable worker execution. The substrate retains full backward compatibility while demonstrating genuine adaptive behavior: in a 20-episode run, the system independently derived two persistent constraints from failure-pattern analysis.

As AI systems become more autonomous, governed cognition emerges as a foundational requirement for trustworthy deployment.

References

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[2] Bernard J. Baars. *A Cognitive Theory of Consciousness*. Cambridge University Press, 1988.

[3] Stanislas Dehaene, Hakwan Lau, and Sid Kouider. What is consciousness, and could machines have it? *Science*, 358(6362):486–492, 2017.

[4] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–253, 1980.

[5] Artur d'Avila Garcez, Luis C. Lamb, and Dov M. Gabbay. *Neural-Symbolic Learning and Reasoning*. Springer, 2019.

[6] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[7] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

[8] Yann LeCun. A path towards autonomous machine intelligence. Technical report, Meta AI, 2022. Version 0.9.2.

[9] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011.

[10] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023.

[11] Gary Marcus. The next decade in AI: Four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

[12] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[13] OpenAI. Function calling and tool use in GPT models. https://platform.openai.com/docs/guides/function-calling, 2023. Accessed: 2024-01-15.

[14] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[15] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.