Horizon Kernel

A simulation-first, verifiable execution system for AI that is beyond 'agentic'

Version: 1.0 **Date:** 14 Feb 2026

One-sentence idea: Put a *compiler + world model simulator + policy verifier* between any AI model and the real world, so the model must **predict** and **prove** what will change before anything executes.

Why a new system is needed

Modern AI agents can talk, plan, and call tools, but they still fail in the real world because they do not reliably model hidden state, side effects, permissions, and drift. Benchmarks and industry reports repeatedly show low first-try success on multi-step, real-world tasks, which blocks adoption in high-stakes workflows.

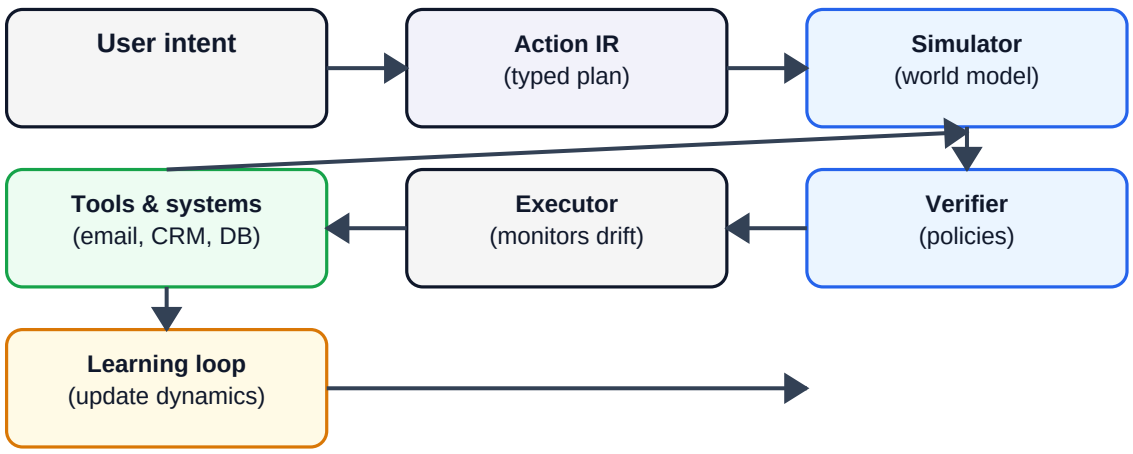
- Agents tend to **act first** and then recover when the environment surprises them.
- Tool ecosystems are **open-world**: new states appear, APIs change, and humans intervene.
- Safety today is mostly UI prompts; what is missing is an **enforcement runtime** that can refuse unsafe or low-confidence actions.

Core principle

simulate - verify - execute, with receipts. Every action must carry a machine-checkable bundle: **preconditions**, **predicted effects** (a diff), **risk score**, and a **rollback plan**. If reality deviates, the runtime halts and replans.

System overview

simulate -> verify -> execute (with receipts)



Horizon Kernel architecture

Think of this as an operating system for AI actions. The model produces proposals; the kernel decides what is allowed and what is likely to succeed.

1) Action compiler (Intent -> Action IR)

Convert natural language goals into a typed intermediate representation (IR) that is explicit about constraints, resources, and side effects. The IR is closer to a 'program' than a chat plan.

- **Typed actions**: send_email(), create_invoice(), update_crm_field(), etc.
- **Constraints**: budgets, deadlines, allowed recipients/domains, required approvals.
- **Required evidence**: which observations must be true before a step can run.

2) State ledger (the digital twin)

Maintain an event-sourced snapshot of the world state relevant to tasks (accounts, files, records, permissions, quotas). Every observation and action is logged to support audit and rollback.

- Unified 'state vector' per integration: last-known values + freshness + authority.
- Immutable log + derived current state (like git + working tree).
- Data minimization: store only what is needed; encrypt and partition by tenant.

3) Simulator (world model for software)

Run the Action IR in 'imagination' before real execution. Predict effects, detect missing preconditions, and explore branches. World-model research shows that precondition and effect prediction can be learned and used for planning.

- Dry-run APIs where available; sandbox accounts otherwise.
- Effect prediction: expected diffs with confidence intervals.
- Uncertainty handling: ask user, gather more observations, or refuse.

4) Verifier (policy + safety)

A deterministic rules engine that can veto, gate, or require approvals. This is where you encode 'never do X' regardless of what the model wants.

- Role-based permissions, action budgets, and rate limits.
- High-risk checks: money movement, external messages, data export, deletions.
- Proof-carrying actions: execution only if preconditions hold and diff is acceptable.

5) Executor (drift-aware runtime)

Execute step-by-step while continually checking that reality matches the simulator. If drift occurs, stop and replan.

- Idempotency keys and retries.
- Rollback or compensating actions when possible.
- Receipts: who/what/when/why, plus before/after diffs.

What makes this beyond 'agentic'

An agent is usually a loop. A kernel is a **control system**. The difference is enforcement:

Typical agent	Horizon Kernel
Acts, then checks	Simulates, then executes
Plans in natural language	Compiles to typed Action IR
Unclear side effects	Diff-first: predicted and audited changes
Safety = prompts	Safety = runtime enforcement + policies
Brittle to drift	Drift-aware execution + re-planning

Fastest path to a real product

Do not start general. Start with one domain where correctness matters and the state space is manageable (e.g., finance ops, revenue ops, security ops). Win by being the first system that is boringly reliable.

90-day build plan

- **Week 1-2:** define 50-100 atomic actions + expected diffs; build Action IR + logging.
- **Week 3-6:** connectors + state ledger + dry-run/sandbox layer; ship diff preview UI.
- **Week 7-10:** verifier policies + approval workflow; idempotent executor.
- **Week 11-13:** simulator learning loop from traces; measure diff accuracy and success rate.

KPIs (the metrics that prove you are winning)

- **Diff match rate:** % of steps where predicted diff equals actual diff.
- **First-try task success:** end-to-end completion without human rescue.
- **Safe-stop rate:** how often the runtime halts before damage (good if it prevents harm).
- **Time-to-recover:** when drift happens, how fast to replan and finish.

References

- Model Context Protocol: Tools documentation (tool annotations such as readOnlyHint/destructiveHint). modelcontextprotocol.io, accessed Feb 14, 2026.
- Model Context Protocol: Changelog (2025-03-26) describing tool annotations and OAuth-based authorization updates. modelcontextprotocol.io, accessed Feb 14, 2026.
- Model Context Protocol: Server tools specification (2025-11-25) noting that tool annotations must be treated as untrusted unless from trusted servers. modelcontextprotocol.io, accessed Feb 14, 2026.
- J. Li et al. (2024). 'Making Large Language Models into World Models with Precondition and Effect Knowledge.' arXiv:2409.12278.
- The Guardian (Aug 5, 2025). Reporting on DeepMind's 'world model' direction for simulation-based decision-making.