Tools Holding an LLM

Haygent deterministic spines, bounded intelligence slots, evidence gates, and truthful halt

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

Tools Holding an LLM

Abstract

Autonomous-agent systems commonly ask a language model to choose the next action, maintain memory, interpret tool output, decide when it is finished, and narrate success. That arrangement gives the least deterministic component responsibility for the system's most consequential properties.

Haygent reverses the architecture. A Haygent is a deterministic, contract-bound process that contains narrow intelligence slots. The runtime—not the model—owns control flow, permissions, state packets, evidence gates, repair budgets, replay, and completion. The model may adjust only declared schema fields. Progress exists only when evidence objects satisfy a gate, and exhausted repair produces a truthful `HALT` rather than a fabricated success.

The concise doctrine is:

Standard agents are LLMs holding tools. Haygents are tools holding an LLM.

Haygent is complementary to the Actor–Agent Runtime. Actor–Agent explains persistent person-shaped state and temporary reasoning. Haygent explains bounded work execution. CAS-native coordination supplies exact inputs and lineage. LCSM supplies accepted state and receipts.

1. The control inversion

A conventional agent loop often resembles:

```
model reads history
→ model chooses a tool
→ model interprets output
→ model decides what to do next
→ model says it is complete
```

Even when a framework wraps this loop, the model frequently retains practical control over sequencing, scope, and completion. A prompt may state permissions, but the runtime may not enforce them. Memory may be a transcript or vector index. “Done” may be generated language rather than an independently evaluated condition.

Haygent begins with the opposite presumption:

```
deterministic spine selects step
→ runtime validates input packet
→ ordinary code performs exact work
→ declared intelligence slot proposes bounded fields
→ runtime validates the proposal
→ gate evaluates evidence
→ spine advances, repairs, or halts
```

Intelligence is a tunable dependency injected at named points. It is not the process itself.

2. The on-disk system is the agent

A Haygent has no legitimate hidden operational identity beyond its versioned package and run ledger.

```
haygents/<slug>/
  haygent.manifest.json
  permissions.json
  contracts/
    input.schema.json
    output.schema.json
    adjustable.schema.json
  spine/
    spine.json
    steps/
  gates/
    gates.json
  prompts/
    adjust.system.md
  score/
    rubric.json
  state/
```

```
    ledger.sqlite
    runs/<run_id>/
  UPGRADE.md
```

The manifest names the job, version, stakes, repair budget, and dependencies. Permissions are runtime grants, not prose. Contracts define what may enter, what may leave, and the only fields the model may change. The spine defines order. Gates define evidence requirements. The ledger records packets, actions, failures, and receipts.

This makes a Haygent inspectable before it runs and replayable after it runs.

3. Immutable packets and exact state

Each transition consumes a packet and emits another packet. Packets are validated, canonically serialized, and hash chained.

A minimal transition record includes:

```
run_id
step_id
input_packet_digest
output_packet_digest
code_version
model_or_adapter_ref?
permission_snapshot
started_at
finished_at
status
evidence_refs[]
prior_transition_digest
transition_digest
```

An intelligence slot cannot mutate the prior packet. It proposes a patch against a declared adjustable schema. The runtime either accepts the patch, requests bounded repair, or rejects it.

Because packets are immutable, reruns can distinguish:

- identical inputs under identical code;
- a model or prompt change;
- a permission change;
- a dependency change;
- a human override;
- nondeterministic output under a pinned contract.

4. Permissions are executable

Haygent permissions use deny-by-default enforcement. A job declares exact capabilities such as:

```
read:file:project/**
write:file:generated/**
run:command:npm-test
network:none
secret:none
publish:none
spend:none
```

The runtime shim checks every requested operation. A model cannot extend its own grant, reinterpret a path wildcard, substitute a tool, or turn a read capability into a write capability.

A permission failure is a first-class result. The correct response is not to improvise around it. The job records the denied operation and either takes a declared fallback or halts.

5. Intelligence slots

An intelligence slot is appropriate where exact code cannot cheaply resolve a bounded ambiguity: classifying an input, selecting from approved candidates, revising a small text field, ranking risks, or proposing a repair.

A slot declares:

```
slot_id
input_projection
adjustable_schema
model_profile
prompt_digest
max_tokens
repair_limit
verifier
fallback
```

The slot does not receive the entire run directory by default. It receives a scoped projection. Its output is not trusted merely because it is valid JSON. The verifier checks semantic constraints, exact references, prohibited substitutions, and evidence.

6. Gates evaluate evidence, not narration

A gate is a predicate over typed evidence. Examples:

```
BUILD_PASS requires exit_code == 0
TEST_PASS requires suite total > 0 and failed == 0
EXACT_ROUND_TRIP requires digest(input) == digest(decoded)
BROWSER_PASS requires named routes and zero critical console errors
HUMAN_ACCEPTANCE requires an authorized human event
```

A model cannot satisfy `TEST_PASS` by saying “all tests passed.” It must deposit the machine-readable test artifact the gate expects.

Gates that cannot fail are not gates. A gate must define invalid evidence, timeout, missing evidence, stale evidence, and contradictory evidence.

7. Bounded repair and truthful HALT

A failed gate may activate a declared repair loop:

```
failure evidence
→ repair projection
→ bounded adjustment
→ rerun affected steps
→ reevaluate gate
```

The repair limit is part of the manifest. The model cannot reset the counter or widen the task. When the budget is exhausted, the Haygent emits:

```
status: HALT
result: FAIL
failed_gate
attempts
last_evidence
preserved_outputs
safe_restart_point
```

HALT is honorable. It protects the project from the much more expensive failure of plausible but false completion.

8. Self-reporting derived from the ledger

A human report and machine score are generated from the same ledger. The report cannot state anything unsupported by events and evidence.

A final Return should include:

- objective;
- input identity;
- permissions used and denied;
- steps executed;
- intelligence slots invoked;
- evidence collected;
- gates passed or failed;
- repair attempts;
- files or artifacts produced;
- limitations;
- receipt and replay command.

Narrative is a projection over the run, not a parallel truth channel.

9. Composition

Haygents compose through typed packets rather than free-form agent conversation.

```
Haygent A Return
→ schema validation
→ authority and scope check
→ exact content address
→ Haygent B input
```

A coordinator may schedule several Haygents, but it cannot merge their outputs by prose alone. Conflicts become explicit objects. A downstream Haygent can require an accepted upstream receipt rather than a completion assertion.

10. Relationship to the wider Glyphd architecture

- **LCSM / Project Core** owns accepted project truth and event history.
- **CAS-native coordination** supplies exact Work Orders, inputs, dependencies, and Returns.
- **Haygent** executes one bounded deterministic work process.
- **Actor–Agent Runtime** maintains durable actors and temporary cognitive processes.
- **Gantry** enforces external crossings.
- **SurfaceMind** renders run state and evidence as an operational surface.
- **Receipts** prove transitions; human acceptance remains separate.

A Haygent is therefore neither a persona nor a memory store. It is an evidence-gated execution unit.

11. Security analysis

Primary threats include prompt injection in inputs, authority spoofing, malicious tool output, path traversal, stale approvals, replay, secret leakage, recursive repair, evaluator compromise, and report/ledger divergence.

Required controls include:

- canonical input digests;
- explicit trust zones;
- non-self-expanding permissions;
- target jail and path normalization;
- endpoint allowlists;
- secret redaction;
- idempotency keys;
- approval expiry;
- immutable packets;
- bounded graph and repair depth;
- independent gate code;
- receipt-chain verification.

A schema is not a security boundary unless the runtime enforces the consequences of the schema.

12. Evaluation

The decisive comparison is not whether a Haygent writes more fluent prose than a free agent. It is whether it completes bounded work more dependably.

Measure:

- unauthorized operation rate;
- invalid packet rate;
- evidence completeness;
- false PASS rate;
- repair convergence;
- replay agreement;
- cross-project leakage;
- time to verified result;
- human intervention count;
- useful HALT quality;
- overhead relative to a conventional agent.

The zero-tolerance metric is false PASS on consequential work.

13. Current evidence state

The Haygent corpus provides a detailed normative specification and draws on implemented mechanics in MegaForm, Glopper, receipt systems, and contract-gated build processes. This paper does not claim that a complete universal Haygent runtime has passed the full benchmark matrix.

The public status is `SPECIFIED` with implemented donor mechanisms. A production claim requires an exact runtime, conformance suite, and result receipts.

14. Limitations

Deterministic spines can be rigid. Poor contracts can prevent useful adaptation. Gate design is labor. Evidence collection has cost. Some tasks are too open-ended for a bounded job and should remain exploratory Chat or planning work. A Haygent can still contain buggy exact code or a flawed verifier.

The architecture does not remove uncertainty. It prevents uncertainty from silently masquerading as authority and completion.

15. Conclusion

Haygent treats intelligence as a component with a declared socket. It gives ordinary software responsibility for sequence, authority, state, evidence, repair, and completion, while retaining language models where bounded judgment is useful.

The result is less theatrical autonomy and more operational agency: a process that can act, explain, fail, repair, replay, and stop without confusing generated confidence with completed work.

Source register

- `HAYGENT-SPEC-v1.0.md`
- `GLYPH-DESK-MASTER-SPEC.v1.md`
- `ZEKE-V1-CANON-RECOVERY-AND-GAP-AUDIT-v0.1.md`
- GlyphCAS design report
- MegaForm and Glopper evidence referenced by the canon recovery

Revision history

- **0.1.0 — 2026-07-27:** Initial public-safe institute edition.