Harness-First Agentic AI

Control Plane, Execution, and Governed Autonomy

AAIAAS™ architecture whitepaper

Version 1.6

October 2026

Table of Contents

1. Executive Summary

2. The Problem: Model-Centric Failure Modes

- 2.1 Three Recurring Failure Patterns
- 2.2 Why the Model-Centric Approach Fails
- 2.3 Inference as Commodity, Tenant State as Value
- 2.4 Production Default: Mid-Size Models + Harness
- 2.4.1 Inference Adequacy
- 2.4.2 Practical Context Size for Local Inference
- 2.4.3 Capacity Placement by Plane
- 2.4.4 Control-Plane Headroom
- 2.5 Limits and Chunking under the Harness
- 2.6 Local Inference Security Boundary

3. Reference Architecture

- 3.1 Six Layers, Three Planes
- 3.2 The Harness Layer: Core Invariants
- 3.3 Execution Plane Invariants
- 3.4 Tenant Isolation

4. Control Plane and Worker Governance

- 4.1 Roles and Responsibilities
- 4.2 Governance Controls
- 4.3 Telemetry and Observability
- 4.4 Managed Concurrency and Admission Control

5. HITL Governance: Risk-Tiered Human-in-the-Loop

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

- 5.1 Risk Tiers
- 5.2 Operation Permanence Classification
- 5.3 Waiver Policy
- 5.4 High-Risk Operation Catalog
- 5.5 HITL as Operator Compounding, Not Only Approval

6. The OGACS™ Policy: Execution Invariants and Operational Governance

- 6.1 What Is OGACS?
- 6.2 Enforcement Model
- 6.3 Invariant Set
- 6.4 Drift and Convergence

7. Identity & Zero-Trust Alignment

- 7.1 Dual-Plane Architecture
- 7.2 Identity & SSO Principles
- 7.3 Zero Trust Principles
- 7.4 Explicit Non-Claims

8. Cognitive Compounding: Reviewed Skill Improvement

- 8.1 The Pattern
- 8.2 Why This Matters

9. Execution Planes and Air-Gap Deployment

- 9.1 Cloud Execution: Multi-Tenant Worker Infrastructure
- 9.2 Device Execution: Tenant-Local Workers
- 9.3 Air-Gap Deployment
- 9.4 Hybrid Configurations
- 9.5 Content Residency by Mode
- 9.6 Secret-Bearing Operations

10. Deployment Topologies and Production Posture

- 10.1 Canonical Deployment Reference

11. Conclusion

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

1. Executive Summary

Peregrines is the platform built on the AAIAAS™ architecture, governed by the OGACS™ policy. This paper specifies the architecture, not a product configuration.

The prevailing narrative in enterprise AI focuses on models: which foundation model is best, which provider offers the best price-performance ratio, which prompt engineering technique unlocks the highest accuracy. This model-centric view produces systems that are fragile, opaque, and difficult to govern. When the model changes, and it will change, the entire system must be rebuilt.

The alternative is to put the harness, the layer that isolates models and governs their use, at the center. The AAIAAS™ reference architecture defines how governed autonomy is built: a control plane that decides, a harness that verifies, and human oversight scaled to risk. The PEREGRINES™ platform realizes that vision with governed workers, each carrying its own skills, operating wherever the work is, from cloud to device to air-gapped site.

This whitepaper defines the **AAIAAS™ reference architecture**: a harness-first control plane for governed autonomy, in which models are interchangeable inputs, planning, policy, verification and proof stay stable, and execution runs provider-agnostic across cloud and device planes, including air-gapped deployment.

The architecture defines five capabilities:

1. **Provider-agnostic execution**: swap models without rewriting task logic
2. **Governed autonomy**: human-in-the-loop gates that scale with managed concurrency, rather than unmanaged agent sprawl
3. **Execution invariants**: circuit breakers, cost limits, and step ceilings that prevent runaway behavior
4. **Proof-of-execution**: every task produces proof artifacts including execution metadata, artifact hashes, and verification results
5. **Cognitive compounding**: execution signals drive proposed skill changes that operators approve and version

This is the reference default deployment. The platform is deployed across one control plane and two execution planes (cloud-shared, device-local), with air-gapped deployment as a mode of the device plane.

Human approval scales with risk: four tiers and a reversible/irreversible classification decide what runs automatically, what notifies an operator and what waits for approval (§5). On identity, workers act inside the customer's own authenticated session or with vault-brokered credentials; the control plane never holds customer SSO secrets (§7).

A key insight from plant-ops work in October 2026: inference adequacy is capability, context, and speed. Capability is a measured pass rate at a context length on a machine, and that rate changes with context (§2.4.1).

This paper defines architectural invariants and the governance model, not product configuration.

Peregrines AI has two U.S. provisional patent applications pending covering governed dispatch of AI agent tasks and boundary lockdown.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

2. The Problem: Model-Centric Failure Modes

2.1 Three Recurring Failure Patterns

Organizations that build autonomous AI systems around model capabilities, rather than around governance and control, encounter three predictable failure patterns:

Pattern A: Manual triggers. Recurring tasks, such as weekly reports, pre-audit evidence pulls or pre-standup checks, run only when someone remembers to start them. When they forget, the downstream process degrades without any signal.

Pattern B: Silent Degradation. Automated processes that appear to function but whose outputs gradually drift from acceptable quality. No alert fires. No metric crosses a threshold. The degradation accumulates until something downstream breaks, by which point hours or days of degraded output have to be traced and redone.

Pattern C: Talent Waste. Engineers and operators spending time maintaining brittle scripts, monitoring broken automations, and manually extracting compliance evidence. This is work that is well defined and repeatable, and could run as a governed autonomous task instead of hand-maintained glue.

These are common problems. Solving them takes better control, not better models.

2.2 Why the Model-Centric Approach Fails

The dominant paradigm treats the model as the product. Engineering effort flows into prompt engineering, model selection, context optimization, and output refinement. Governance is bolted on after the fact as a secondary concern.

This produces fragile systems because:

- **Model swap requires rewriting.** When a provider is deprecated, pricing changes, or a better model emerges, the task logic must be re-engineered because execution and model coupling are inseparable.
- **Governance is inconsistent.** Ad hoc safety checks and manual approvals become inconsistent as volume grows, which is when consistency matters most.
- **Audit trails are incomplete.** When every execution path is model-dependent and ad hoc, producing proof-of-execution for compliance is an engineering project, not a native capability.

The harness-first architecture inverts this priority. The model is a commodity input; the harness is where the value sits.

2.3 Inference as Commodity, Tenant State as Value

Inference is a swappable input: the harness lets you change model family, provider or endpoint without rewriting task logic. What differentiates one deployment from another is the tenant-scoped state each worker builds over time: memory artifacts, reviewed skills, policy-bound history and execution records. Two tenants on the same model and harness will get different outcomes because that state differs. The design expectation is that return on investment depends more on how well that state captures the tenant's operations than on which model is chosen.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

2.4 Production Default: Mid-Size Models + Harness

Most production agent workloads sit in the body of the difficulty distribution: retrieval and synthesis, structured extraction, scoped tool use, short-horizon planning, and verification against explicit contracts. For these tasks, a well-trained mid-size model (tens of billions of parameters) is often sufficient; on these tasks, further scale tends to add latency and cost faster than quality.

The harness—covering planning, policy, admission control, verification before completion, and proof of execution—determines reliability more than raw parameter count. Provider-agnostic design routes bulk work to efficient mid-size models by default. Stronger models remain available where the architecture places them, primarily with control-plane orchestration, not as the volume path for every worker turn.

> *For production volume, a strong mid-size model behind a real control plane is the rational default. Capability is placed by plane: efficient mid-size inference for workers; control-plane headroom for governance and the rare cases that need it.*

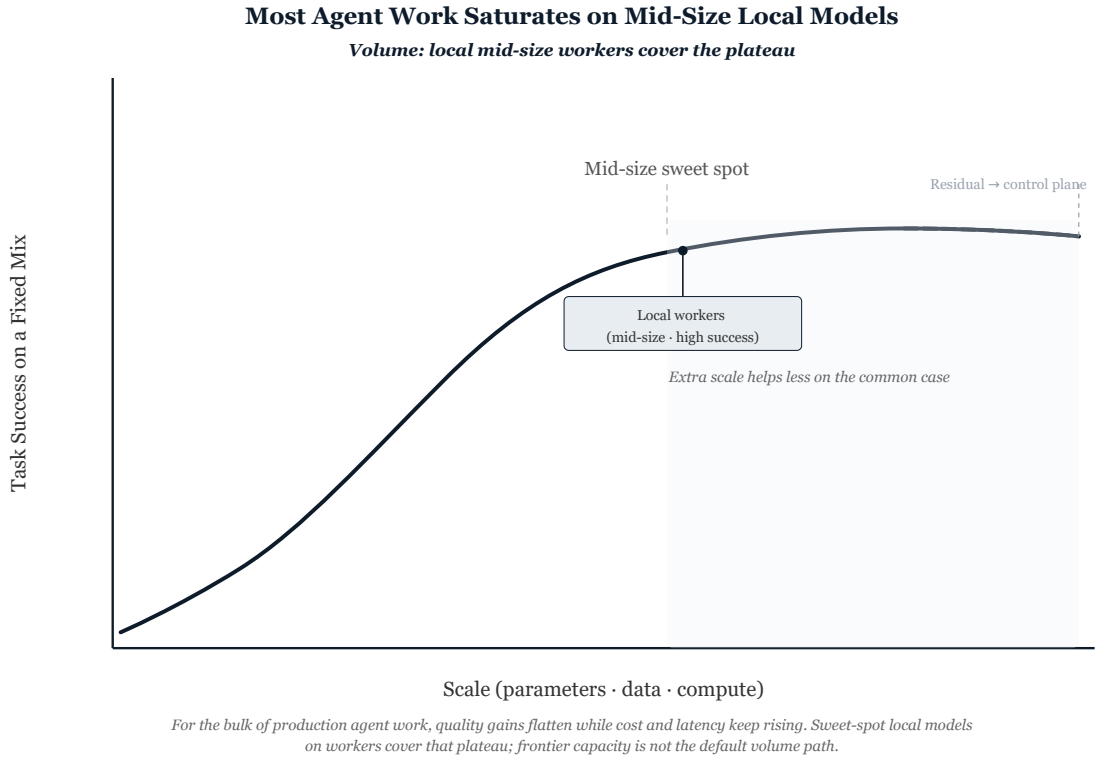


Figure 1. Diminishing returns on a fixed task mix: task success rises with scale, then flattens. Mid-size workers operate on that plateau; frontier capacity is not the default volume path.

Most work sits on the plateau, where mid-size models suffice; stronger capacity is held at the control plane for the few steps that need it.

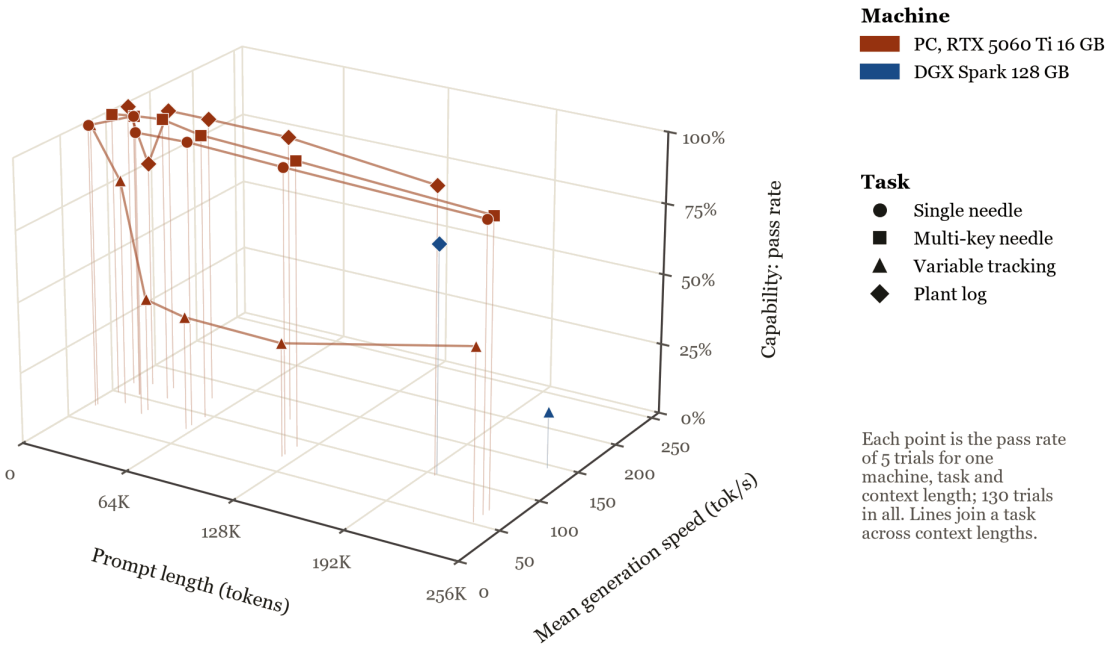
 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

2.4.1 Inference Adequacy

Inference adequacy is determined by CCS: capability, context, and speed. Capability is a measured pass rate per context length and per machine, and it changes with context. Context is the working set the model holds. Speed is generation rate in the operator's loop. Figure 2 plots prompt length, mean generation speed, and that pass rate. One point is one machine, one task, and one context length.

> *The brain in AI is only as large as the context it can hold.*
> *— Dr. Guy Savage*

Capability, context length, and generation speed



Source: Peregrines bench, October 2026.

Figure 2. Capability, context length, and generation speed. Each point is the pass rate for one machine, one task, and one context length. The DGX Spark points are a long-context spot check, 2 tasks, n=5 each.

Cold-start latency is an engineering variable the control plane can manage, for example by keeping models warm. Measured 256K results are in §2.4.2 and Figure 3.

2.4.2 Practical Context Size for Local Inference

Measured 256K (262,144 token) context data on two hardware platforms:

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

Table 1. Hardware · Quantization · Context · Prompt · Output · Runs

Hardware	Quantization	Context	Prompt	Output	Runs
DGX Spark (128 GB unified memory)	nvfp4 (qwen36-35b-a3b-nvfp4)	262144	245878	12	1
PC with RTX 5060 Ti (16 GB)	iq2_xs (qwen3.8-flash-next), KV q4_0	262144	245878	12	1

Note: at 256K context the DGX Spark ran Qwen3.6-35B-A3B (NVFP4) for every score reported here; the PC with RTX 5060 Ti ran Qwen3.8-Flash-Next (IQ2_XS). Prompt and Output are the medians of prompt_tokens and completion_tokens across the 20 trials in that one run (5 trials on each of 4 tasks).

Needles, planted facts, variable tracking, generation speed, and cold TTFT are reported from those runs. The DGX Spark (128 GB unified memory) scored 10/10 needles, 4/5 planted facts, 1/5 variable tracking, ~109 tok/s generation, and ~359 s cold TTFT. The PC with RTX 5060 Ti (16 GB) scored 10/10 needles, 5/5 planted facts, 3/5 variable tracking, 67.8 tok/s median generation, and 174.5 s median TTFT.

Multi-seat capacity on the DGX Spark (128 GB) is a theoretical projection based on its unified-memory headroom; it has not yet been load-tested. Concurrency benchmarking is planned.

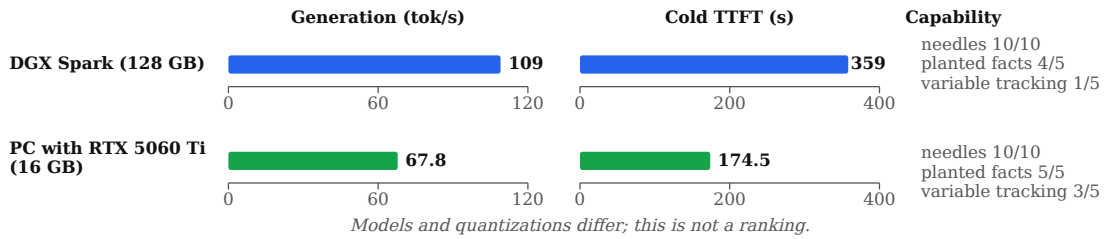


Figure 3. Two-panel comparison at 256K context: generation speed (tok/s) and cold TTFT (s). Both systems scored on needle retrieval, planted facts and variable tracking; both ran the same five planted-facts items at 256K.

Models and quantizations differ, so this is not a ranking. Cold time-to-first-token at 256K (about 3 to 6 minutes) is too slow for an interactive operator loop. Results were not filtered by a TTFT threshold.

2.4.3 Capacity Placement by Plane

Capability is **placed by plane**, not treated as a temporary compromise.

- **The Peregrines platform** (execution volume): efficient mid-size inference on the body of the task distribution.
- **Cloud control plane** (governance): planning, admission, policy, and the rare steps that need more reasoning headroom.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

For the body of production agent work, quality gains from further model scale diminish while cost and latency keep rising. The rational operating point sits on that plateau. Under SQDEC (Safety > Quality > Delivery > Economics > Control), that is efficient default placement: Safety and Quality set the bar; Economics and Delivery improve when volume stays on the plateau.

> *Past the body plateau, further scale raises cost and latency faster than it raises success on typical workloads.*

Under SQDEC, routing volume work to frontier-scale models over-provisions capacity: Safety and Quality are already met on the plateau; Economics and Delivery degrade when every worker pays for headroom it does not use.

2.4.4 Control-Plane Headroom

When a step needs more headroom after decomposition and verification, stronger models assist orchestration and hard judgment at the control plane. Workers stay on efficient mid-size inference. There is no silent always-on high-scale path on every worker turn; headroom is explicit, gated, and rare relative to volume.

2.5 Limits and Chunking under the Harness

Mid-size models still underperform on long-horizon, weakly decomposable, or rare-knowledge work. The architecture does not claim otherwise. The harness addresses those limits by decomposing large outcomes into short, verifiable units that a mid-size model can handle, with each unit verified before the next begins.

Chunking is a control-plane move: steps that remain hard after honest decomposition escalate to stronger inference at the control plane, not to every worker.

2.6 Local Inference Security Boundary

When workloads use device-local, on-premises, or air-gap inference endpoints, sensitive content can be processed without sending that content to external cloud model providers. This is a capability of the deployment topology, not a property of the harness itself.

---

3. Reference Architecture

3.1 Six Layers, Three Planes

The AAIAAS™ architecture is organized into six distinct layers, each with a bounded responsibility. This layering is not optional. Violating layer boundaries requires an architectural decision record (Change Record) and explicit review.

Table 2. Layer · Responsibility

Layer	Responsibility
Experience Layer	User interfaces, dashboards, input capture
Orchestration Layer	Planning, routing, policy enforcement, task decomposition
Harness Layer	Model/tool abstraction, provider isolation, verification, governance
Capability Layer	Modular skills and connectors; provider-agnostic interfaces
Execution Layer	Workers that execute tasks on their assigned execution plane
Memory Layer	Tenant-segmented knowledge storage and retrieval

The architecture defines one control plane and two execution planes (cloud-shared and device-local), with air-gap deployment as a mode of the device plane.

Table 3. Plane · Role

Plane	Role
Control Plane	Strategic orchestration, planning, policy, and governance
Cloud Execution Plane	Shared multi-tenant worker infrastructure
Device Execution Plane	Tenant-owned local or on-premises workers

No plane may silently assume the responsibilities of another. The control plane plans and evaluates but does not execute. Execution workers execute but do not plan. This separation is the architectural invariant that makes governed autonomy possible.

3.2 The Harness Layer: Core Invariants

The harness layer is defined by five core invariants, plus four supporting ones:

Invariant 1 — Provider Isolation. Provider-specific execution logic is confined to dedicated adapters. Task logic (skills) never embeds model-provider concerns. Swapping a model provider requires changing the adapter, not rewriting the skill.

Invariant 2 — Provider-Agnostic Capabilities. Skill semantics are defined independently of any model. A "browse the web" skill produces the same output contract whether powered by a commercial hosted model or a self-hosted open-weight model.

Invariant 3 — Graceful Degradation. Optional enrichment steps (retrieval-augmented generation, planner critic passes, billing telemetry) degrade gracefully unless explicitly marked as hard-required by policy. Failures produce structured diagnostics, not cascading failures.

Invariant 4 — Verification Before Completion. Task completion requires satisfaction of verification policies. A task does not complete until its output is verified against the acceptance criteria defined in the task spec.

Invariant 5 — Bounded Execution. Task graph expansion obeys explicit depth and fan-out limits. The system will not generate infinitely recursive task chains. Every execution path has a ceiling.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

The supporting invariants enforce: Safety hierarchy (the safety tier of SQDEC cannot be overridden by delivery or economic pressure); Tenant-segmented memory; Artifact separation (durable artifacts stay external and addressable, not folded into prompt context); Execution capability check (before admitting a task graph, the harness verifies that the target plane can meet each skill's contract).

3.3 Execution Plane Invariants

The execution layer is governed by its own invariant set:

- **Plane Separation:** Control, cloud, and device responsibilities remain clearly delineated. No plane assumes the responsibilities of another.
- **Worker Traceability:** Every worker maintains a verifiable identity including worker ID, execution plane assignment, and authorized tenant set.
- **Proof-of-Execution:** Every completed task emits proof artifacts including execution metadata, artifact hashes, and verification signals.
- **Fail-Closed Execution:** Workers fail closed on unsupported skill types, invalid payload schemas, or missing authorization. Silent fallback execution is forbidden.
- **Tenant Authorization:** Device workers serve a single tenant. Cloud workers serve an authorized set.

3.4 Tenant Isolation

Every tenant undergoes a bootstrap lifecycle before activation: a defined provisioning sequence that establishes tenant record, settings, feature flags, memory namespace, public skill access, execution authorization, and observability scope. A tenant must not reach active status until all bootstrap resources exist. Authentication success alone does not imply readiness.

Tenant data, memory and execution context are partitioned per tenant; the architecture has no path for one tenant's task to read another tenant's data.

---

4. Control Plane and Worker Governance

4.1 Roles and Responsibilities

The platform separates orchestration from execution:

Control-plane orchestrator. Receives intent, expands goals into task specifications, generates execution plans, and exercises human-in-the-loop approval authority. Orchestration logic lives in the control plane, never in the execution layer.

Worker agents. Execute tasks dispatched by the control plane on an assigned execution plane, but act only under control-plane direction. A worker that cannot report status or accept recall is treated as a system failure.

Staging environment. Local development and staging is where skills are developed and configurations tested before production deployment.

Operator dashboard. Where operators monitor task state, work the HITL approval queue and direct execution.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

> ***Role definitions.*** *The component that decides is the control plane (orchestrator); the component that runs a task on an assigned execution plane is the worker (executor); the capability a worker invokes is a skill. This paper uses these functional terms throughout. www.peregrines.ai is the product surface; this whitepaper remains the reference document.*

4.2 Governance Controls

Three mechanisms bound autonomous execution:

HITL gate. Tasks at the High and Critical risk tiers are held behind a human approval gate. No task bypasses this silently. Approval is never removed by default; it requires explicit human action (approve, reject, or waive-by-action at the individual task level).

Budget constraints. Every worker has an associated cost envelope. Execution paths are bounded by per-task and per-session spending limits. When the envelope is exhausted, execution halts and reports; it does not continue silently.

Control-plane tether. Workers remain connected to the control plane through heartbeat signals and recall paths. Even workers running locally with local inference assistance are tethered to the central orchestrator. They can operate semi-independently but can always be recalled.

4.3 Telemetry and Observability

Every worker emits telemetry at every lifecycle stage, covering planning, dispatch, execution, completion, and verification. Silent operation is not an option. Every task produces structured logs suitable for aggregation and analysis, with traceability across tenant, worker, execution plane, and skill.

4.4 Managed Concurrency and Admission Control

Workers may execute in parallel as a capacity optimization, but parallelism is never an unmanaged default. The control plane exercises authoritative task admission and sequencing. Unbounded multi-agent execution without admission control is a governance failure mode, producing conflicting writes, unrecoverable state, and unauditable outcomes rather than operational maturity.

Doctrine (reference):

- Tasks are admitted by the orchestrator; workers do not self-dispatch or peer-schedule as orchestrators.
- Parallel workers are optional capacity under control-plane direction, not independent planners.
- Serial-first is the safe default; parallelism is introduced only when shared-state contention is controlled.

This is the difference between a choreographed fleet and agent sprawl, with one orchestrator directing traffic under governance. Exact WIP bounds, admission algorithms, and deployment knobs are defined in operating guides and product configuration, not in this reference architecture.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

5. HITL Governance: Risk-Tiered Human-in-the-Loop

5.1 Risk Tiers

The platform classifies every operation into one of four risk tiers, and each tier has a defined approval requirement:

Table 4. Tier · Definition · Default Action

Tier	Definition	Default Action
Low	Routine, idempotent, or read-only operations with no security or cost impact	Auto-approve
Medium	State-changing operations with limited scope or reversible impact	Notify only
High	Critical state changes, significant cost implications, or security-sensitive actions	Require approval
Critical	Irreversible destructive actions or root-level security changes	Require typed confirmation

5.2 Operation Permanence Classification

Operations are further classified by permanence:

- **Type 1 (Reversible):** Operations where the system state can be restored via automated rollback or point-in-time recovery. These require a documented rollback plan in the task metadata but may proceed under medium-tier rules.
- **Type 2 (Irreversible):** Operations that result in data loss, irreversible state changes, or non-refundable costs. These always require HITL approval regardless of risk score and must include explicit risk acceptance.

5.3 Waiver Policy

Waivers are granted per-action only. No session-wide bypass is permitted. No silent bypass exists; every waived HITL checkpoint generates an auditable event. System-level invariants, specifically the safety constraints of the SQDEC ordering, cannot be waived under any circumstances.

High and Critical risk operations cannot be approved via voice interfaces. Critical risk operations require typed confirmation to prevent accidental activation.

5.4 High-Risk Operation Catalog

The following operation categories are hard-coded as High or Critical risk:

- Deploy to Production: **High**
- Delete Data (Database or Storage): **Critical** (Type 2, Irreversible)
- Billing Changes (Plan upgrade, downgrade, or cancellation): **High**
- Secret Rotation or Credential Invalidation: **High**
- Skill Promotion (Development to Production): **High**

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

This catalog is not exhaustive; any new operation category is evaluated against the same risk framework, but these are the established baseline for the platform's operating posture.

5.5 HITL as Operator Compounding, Not Only Approval

Risk-tiered approval is necessary but not the whole job. HITL also keeps the operator learning. Agentic tools make competent operators faster, through more reuse of proven skills and less repetitive work, but they don't remove the need for an operator or make undisciplined operation adequate. Judgment about scope, risk and 'good enough' stays human. That judgment compounds only when people stay engaged: approving high-stakes actions, reviewing proof of execution and outstanding work, and periodically redesigning workflows. Set-and-forget removes the corrective signal for both the agent and the operator. This doesn't mean approving every step. Risk tiers still route routine work to auto-approve or notify. It means meaningful review stays available, and is used, where quality, scope and irreversibility call for it.

> *There's no magic bullet; the operator still has to know what they're doing.*
>
> *— Dr. Guy Savage*

---

6. The OGACS™ Policy: Execution Invariants and Operational Governance

6.1 What Is OGACS?

The OGACS™ governance policy applies across the layers. It defines the policy stack, covering SQDEC priority order, Change Record (CR) review gate, verified completion evidence, HITL gating, vault governance, and doctrine, that autonomous systems follow.

Products built on this architecture apply OGACS policy within a dual-plane architecture; this paper specifies the policy and invariants, not a single product configuration.

The OGACS™ policy is enforced as an invariant layer with a fixed, auditable set of constraints, not a dynamically loaded rule engine. Its role is to prevent drift, which is divergence between intended and actual execution behavior.

For correspondence regarding the OGACS™ policy or the AAIAAS™ reference architecture, contact info@peregrines.ai.

6.2 Enforcement Model

The OGACS™ policy evaluates operations through two primary dimensions:

Operational Mode. The system operates in one of two modes:

- **OPEN mode** (standard): All capabilities are available, standard governance applies. This is the default operational condition.
- **LOCKDOWN mode** (containment): Execution is suppressed; only safety-critical operations are permitted, and all other operations require explicit operator authorization. Skill loading collapses to the safety layer only.

 • PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

Drift Detection. The system compares operational state against an authoritative baseline. When drift is detected, the OGACS policy requires controlled convergence, automated where safe and escalated to operators where judgment is required, rather than silent divergence.

6.3 Invariant Set

OGACS™ invariants cover the following domains:

- **Approval Gates:** Operations at defined risk thresholds require human authorization before execution. No bypass.
- **Step Limits:** Each execution path has a maximum step count. The system will not execute beyond the limit.
- **Cost Limits:** Per-task and per-session cost ceilings prevent runaway spending.
- **Circuit Breakers:** When error rates, timeout rates, or quality degradation exceed thresholds, the system halts execution and escalates to human review.
- **Tenant Segregation:** Each task runs in exactly one tenant's context. Tenant data, memory and artifacts are partitioned by tenant. A cloud worker may serve several authorized tenants, but never mixes their data in one task. Any operation that spans tenants requires explicit authorization and is logged.
- **Artifact Separation:** Durable outputs remain externalized and addressable. Prompt context and execution memory never replace artifact records.

6.4 Drift and Convergence

The OGACS™ policy treats divergence from the authoritative baseline as a first-class governance event. When drift is detected, the system requires a controlled return to baseline, via automated reconciliation where safe or operator escalation where judgment is required, then verifies that authoritative state is restored.

Drift handling is a continuous architectural property, not a periodic audit exercise. Step-by-step convergence runbooks, severity taxonomies, and merge procedures are maintained in operating guides.

---

7. Identity & Zero-Trust Alignment

7.1 Dual-Plane Architecture

The AAIAAS™ architecture follows a dual-plane identity model: a cloud control plane (policy, audit, orchestration) and a customer- or device-hosted execution plane, where the agent acts. This separation ensures that identity secrets never transit the control plane.

> *A worker runs on the customer's governed endpoint (desktop/virtual desktop infrastructure (VDI)) and drives that session, under the control plane's policy and tether. For interactive SSO applications it operates under the user's (or approved robot's) already-established identity, credential, and access management (ICAM) session, similar to an attended robotic process automation (RPA) agent. Long-lived or API access uses vault-brokered delegated*

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

> *credentials, not password relay through the cloud control plane. The control plane provides policy and audit; it does not hold customer SSO secrets.*

7.2 Identity & SSO Principles

The platform implements two supported SSO access patterns and one forbidden pattern, aligned with Zero Trust and Identity, Credential, and Access Management (ICAM) best practices:

Table 5. Model · How identity is obtained · ZT fit

Model	How identity is obtained	ZT fit
A. Attended / session inheritance	User or robot already authenticated on endpoint (identity, credential, and access management (ICAM), Personal Identity Verification / Common Access Card (PIV/CAC), MFA, virtual desktop infrastructure (VDI)). A worker drives that session.	**Best default**, identity from customer or agency identity provider (IdP)
B. Vaulted app identity	Service account / OAuth client / personal access token (PAT) from customer vault at edge.	Preferred for APIs and unattended work
C. Cloud CP holds SSO	Password relay or SaaS browser as user without endpoint session	**Forbidden**, violates zero-trust

7.3 Zero Trust Principles

Table 6. **ZT Principle** · **Worker Implication**

Never trust the network	Workers on a governed endpoint (desktop / VDI / enclave), not a cloud-hosted browser holding identity, credential, and access management (ICAM) session
Verify explicitly (ICAM)	Identity is issued by agency IdP. A worker does **not** replace identity, credential, and access management (ICAM)
Least privilege	Policy envelope governs which sites, vault paths, and sessions are accessible
No secret sprawl	No SSO passwords in chat; no CP as customer IdP; short-lived tokens preferred
Continuous evaluation	Session expiry or step-up MFA triggers a worker fail-closed response, which escalates to human re-auth

Credentials at the edge. Access credentials are resolved from an approved vault system at the execution plane. The control plane provides policy, audit, and orchestration; it is not a password store. This keeps secrets out of chat logs, prompts, and model context.

No secret sprawl. No long-lived secrets in chat, prompts, or as the model's source of truth for credentials. Short-lived, least-privilege credentials preferred. No model or harness component serves as a de facto credential repository.

A worker operates **in** the governed endpoint (not around the customer's identity, credential, and access management (ICAM)). This is a reference design aligned with Zero Trust principles. It is not

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

a Zero Trust certification, FedRAMP authorization, or identity, credential, and access management (ICAM) replacement.

7.4 Explicit Non-Claims

The following are **not** claims of this design:

- Workers do **not** bypass identity, credential, and access management (ICAM), CAPTCHA, or agency anti-automation by design
- Workers do **not** inherit customer ICAM via a SaaS browser in the PEREGRINES™ platform without an endpoint install
- Workers do **not** accept password paste into chat or use the CP as a customer identity provider
- This whitepaper is not an Authority to Operate (ATO), FedRAMP, or identity, credential, and access management (ICAM) certification document
- Open-web and hostile-UX coverage is capability-scoped; production use is limited to surfaces that satisfy the verification and fail-closed invariants defined herein

---

8. Cognitive Compounding: Reviewed Skill Improvement

8.1 The Pattern

Many agent deployments are static: a skill runs as designed, and improving it means manual re-engineering. Cognitive compounding adds a reviewed improvement loop. Execution produces structured quality signals (completion, verification pass rate, operator feedback, cost and effort), and those signals drive proposed skill changes for operator review. This isn't model training: no weights change, and no unreviewed skill change is promoted.

Reference loop (principle-level):

- **Immediate:** Per-run signals accumulate so operators can see skill health trend, not only pass/fail of the last run.
- **Iterative:** Underperforming skills can enter a reviewed improvement cycle; approved changes are versioned before promotion.
- **Compounding:** Patterns that recur across skills may be abstracted into shared improvements over time, raising the floor for related capabilities.

Exact scoring formulas, health-score UIs, and calendar cadences are defined on the product surface and in operating guides.

8.2 Why This Matters

The cognitive compounding pattern produces a system whose capabilities grow over time without engineering effort growing in step with them. The first run of a skill delivers baseline value. The twentieth run, informed by accumulated execution data, delivers measurably better results. The hundredth run, benefiting from cross-skill patterns, delivers results that no single engineer could have engineered manually, similar to how one developer used AI agents and over 500 billion tokens to rebuild Modern Warfare 2's source code, matching 83% of the original functions byte-for-byte.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

The agents worked in verification loops, checking output against the original binary before accepting it.

That growth depends on continued human engagement in the loop, not only as a safety brake but as the reviewer and redesigner of the skills. Skill proposals require operator review; quality signals include operator feedback; redesign of underperforming workflows is a human act. Set-and-forget removes the corrective channel that keeps both the skill library and the operator's judgment calibrated (see §5.5).

Reviewed improvement is how the architecture aims to grow capability without engineering effort growing in step with it, while keeping a human check so silent degradation can't pass for progress.

---

9. Execution Planes and Air-Gap Deployment

9.1 Cloud Execution: Multi-Tenant Worker Infrastructure

The cloud execution plane provides shared, multi-tenant worker infrastructure hosted on managed cloud services. Workers in this plane:

- Serve multiple tenants simultaneously
- Are managed and upgraded centrally
- Share provider connections; any caching is partitioned per tenant
- Require an authorized tenant set for cross-tenant access
- Always maintain the control-plane tether, providing heartbeat, registration, and recall paths

This is the reference default deployment.

9.2 Device Execution: Tenant-Local Workers

The device execution plane provides tenant-owned local or on-premises workers. Workers in this plane:

- Serve a single tenant exclusively
- Run on infrastructure controlled by the tenant
- May operate with local inference models (Ollama, local LLMs) for preprocessing, retrieval, summarization, and redaction
- Maintain the control-plane tether; they can operate semi-independently with local-assist inference but remain connected to the central orchestrator
- Support hybrid configurations in which local inference preprocesses data for the control plane's planning without taking part in it

Local inference in device mode is explicitly constrained: it operates as a Harness Layer co-processor, not as a second orchestrator. It may assist with local retrieval, summarization, fact extraction, redaction, and privacy-preserving preprocessing. Strategic planning, policy decisions, and authoritative task orchestration remain control-plane responsibilities.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

9.3 Air-Gap Deployment

For tenants with strict data residency or sovereignty requirements, the architecture supports air-gap execution:

- The control plane is deployed inside the same enclave; the tether never leaves it. Telemetry and audit records stay on-premises. Any export crosses the boundary only by the tenant's own approved transfer process.
- Workers execute entirely within the tenant's network boundary
- Model inference runs on local hardware
- All artifacts, proofs, and audit logs are generated and stored on-premises

This is the most restrictive deployment mode and needs careful operational planning; the architecture is designed for it from the start.

9.4 Hybrid Configurations

Many enterprise deployments are expected to be hybrid.

- Strategic planning runs in the cloud control plane
- Data-sensitive preprocessing runs on local workers with local models
- Heavy computation tasks run in the cloud plane
- Compliance-critical verification runs on-premises, where that data lives
- The control-plane tether connects everything, ensuring that even locally operating workers can be directed, recalled, and audited by the central orchestrator

9.5 Content Residency by Mode

Table 7. Mode · Sensitive content to external model API? · Notes

Mode	Sensitive content to external model API?	Notes
Device-local inference endpoint	No (stays on device / local network)	Co-processor; preprocessing, redaction, retrieval
Hybrid	Depends on task path	Cloud control plane / cloud workers may still see admitted payloads
Air-gap	No outbound from execution plane except by tenant-approved transfer	Strictest residency; model runs on local hardware

9.6 Secret-Bearing Operations

Some work handles key material or privileged credentials, for example signing and deploying smart contracts, operating hardware security module (HSM)-backed systems, or rotating production secrets. The first rule is §7's: keys stay in the vault or HSM, and signing runs as a tool call that returns a result, not the key. Where that isn't possible and sensitive material will reach the model's context (tool outputs, logs, reasoning traces), the task runs on customer-hosted inference inside the customer boundary.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

Policy: tasks touching signing keys, seed phrases, or privileged credentials route to local inference only; signing/deploy is Type 2 (irreversible) and always requires pre-approval. For these tasks, local routing is mandatory policy, not a setting.

---

10. Deployment Topologies and Production Posture

10.1 Canonical Deployment Reference

The platform supports the following deployment topologies:

Production Cloud (default). The control plane is deployed on a managed cloud service with a managed database. Workers run in the cloud execution plane. Local device workers are optional. This is the standard configuration for most tenants.

Sovereign/On-Premises. The control plane is deployed within the tenant's infrastructure. Model providers may be external (via secure API) or fully local. Workers run on tenant hardware. This configuration is designed for strict data-residency and sovereignty requirements.

Hybrid. The control plane runs in the cloud. Device workers run on-premises or at the edge. The two communicate over encrypted channels with the control-plane tether, enabling many enterprise deployments to be hybrid.

---

11. Conclusion

The harness-first approach moves the engineering investment from choosing models to governing their use: provider isolation, verification before completion, bounded execution, risk-tiered human approval and proof of execution. Models will keep changing; an architecture that treats them as swappable inputs keeps its task logic, policy and audit trail stable through those changes. On the common case, a mid-size model inside that harness is the practical default, with stronger capacity placed at the control plane where it is needed. This is a production posture, not a stopgap: autonomy that can be trusted, audited and held accountable.

Call to Action

Operators: Check your current agent deployments against four questions: Can you swap the model without touching task logic? Does every task stop at an explicit step, cost and depth limit? Do high-risk and irreversible actions wait for a human? Can you produce proof of what ran?

Architects: Draw the plane boundaries first. Put planning and policy in the control plane, execution in workers, and provider-specific code in adapters, then wire verification and admission control in before the first task runs.

Leaders: Delegate authority with boundaries: risk tiers decide what runs on its own, what notifies you and what waits for approval.

The question is no longer *which model*. It is *what governs it*.

 PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.

This whitepaper is published by Peregrines AI, Inc., which maintains the AAIAAS™ architecture and the OGACS™ governance policy it specifies. It defines the architectural invariants and the governance model. Product-specific actor taxonomies and control-plane details are published on the PEREGRINES™ platform.

Agentic AI with delegated authority, operating wherever needed. Deployed across Cloud, Device, and Air-Gap planes.

Questions? Contact info@peregrines.ai.

PEREGRINES™, AAIAAS™, and OGACS™ are trademarks of Peregrines AI, Inc. All rights reserved.