TECHNICAL REFERENCE ARCHITECTURE

Agentic AI production governance

Pre-action policy checks, authorized human review, and independently verifiable evidence

Document ID VP-ARCH-2026-01 | November 2026 | Version 2.1

For principal architects, staff engineers, and security leaders

Contents

01 Execution state and enterprise governance
02 Integrating a policy check with a workflow
03 Correcting requests during review
04 Evidence packs and independent verification
05 Deployment models and failure handling
06 Architecture and ownership costs
07 Rollout and verification
08 References
09 Appendix: Inside an evidence pack

Executive summary: LangGraph, the OpenAI Agents SDK, and Temporal provide ways to pause work for a human decision and resume it later. For prototypes, internal tools, and low stakes automation, a framework interrupt that posts to a chat channel works well. Once agents execute production transactions, change customer records, or go through a CISO's security review, teams need a clear split between the execution plane and a governance control plane. This paper compares framework-native human-in-the-loop patterns with dedicated runtime governance. It shows how engineering teams pair native interrupts with VeriProof to change policy without redeploying code, enforce role-based authority at decision time, and produce audit evidence an outside party can verify.

1. The maturity spectrum: execution state vs. enterprise governance

When framework-native interrupts are enough, and when a separate governance control plane earns its place.

LangGraph's `interrupt()` saves graph state through a checkpointer such as `PostgresSaver` and stops the run until the application calls `Command(resume=...)`. The OpenAI Agents SDK pauses on tools marked `needs_approval` and resumes from a serialized `RunState`. Temporal workflows wait for an approval signal with `workflow.wait_condition()`. Recovery depends on durable workflow state. LangGraph needs a persistent checkpointer, the OpenAI application must save its RunState, and Temporal records workflow history.

Whether native interrupts are enough depends on system scope and operational risk:

- Stage 1: prototypes, internal tools, and reversible workflows. When one team owns the agent and the review channel, and a mistake is cheap to undo, an interrupt that posts to Slack is a sound, low-overhead design. Drafting content, triaging internal tickets, and developer tooling usually fit here.
- Stage 2: regulated production, high-stakes actions, and multi-team scale. When agents issue refunds, change health records, or move customer money, security and audit teams ask who was allowed to approve, which rule applied, and whether the approved action is the one that ran. Making the execution framework act as identity verifier, policy store, and audit ledger at this scale creates bottlenecks and compliance gaps.

Escalated action: policy check → pause → authorized decision → signature and policy checks → application acts and reports the result

Architectural layer	Native framework primitive	Inflection point at enterprise scale	VeriProof control plane extension
1. Policy evaluation	Thresholds as if/else checks inside tool code.	Coupled code and policy. A limit change takes a pull request, a CI run, and a deploy, with no shadow evaluation first.	Decoupled policy engine. Versioned rules outside agent code. Teams can test a rule in Shadow mode and stage enforcement across selected applications.
2. Identity and authority	An approve button in a shared channel, or a	A shared channel alone cannot establish who has authority to approve the action.	Authority checked at decision time. Only people with review permission for that application can

Architectural layer	Native framework primitive	Inflection point at enterprise scale	VeriProof control plane extension
	resume endpoint any service can call.		decide. SSO sign-in details are recorded.
3. State integrity (TOCTOU)	Checkpoint rows resumed with Command(resume=...).	If the application resumes from editable arguments without checking them, a change after approval can alter what runs.	Signed decision callbacks. An Ed25519-signed callback bound to digests of the payload and review request. The SDK checks both before resuming.
4. Exception UX and NIGO triage	Raw tool arguments posted to a webhook.	A yes-or-no form gives the reviewer no way to correct a single field.	Structured in-flight resolution. Typed, validated review forms. Reviewers correct editable fields or send the application a question.
5. Audit and ledger evidence	Checkpoint blobs plus traces with short retention.	Separate chat, application, and trace records make it harder to reconstruct one decision and check its integrity.	Signed evidence packs. The verifier checks the exported records against the signed inventory and the tenant's ledger.

2. Reference integration: wrapping the interrupt hook

Runtime governance without changing your orchestrator's architecture.

VeriProof doesn't replace the orchestrator. LangGraph, the OpenAI Agents SDK, or your own runtime still owns state, control flow, and tool calls. The SDK wraps each protected action, asks for a policy decision before it runs, and checks the output before the result reaches the caller.

Standardized pre-action outcomes

Every evaluation returns one of five outcomes. When several rules match, the strictest wins, in the order Deny, Escalate, Transform, Warn, Allow.

- Allow: the application may proceed with the proposed action.
- Warn: the application may proceed and records the warning.
- Transform: the application uses the returned replacement for the protected input or output.
- Escalate: the action pauses and a review case opens. The SDK returns a durable checkpoint, which the application stores until a signed decision arrives.
- Deny: the action doesn't run.

Code progression: Stage 1 prototype vs. Stage 2 production

The second example moves the policy check into the refund handler. A protected refund requires a valid policy decision before the application calls the payment service. The first example is a typical LangGraph refund node, and the second governs the same refund with the VeriProof Python SDK.

Stage 1: framework-native approval inside a LangGraph node

```python
from langgraph.types import interrupt
```

```
def issue_refund(state: RefundState) -> dict:
    # The threshold is code: changing it takes a pull request and a deploy.
    if state["amount"] > 500:
        # The application must authenticate whoever resumes the run.
        decision = interrupt({
            "account_id": state["account_id"],
            "amount": state["amount"],
        })
        if not decision.get("approved"):
            return {"status": "rejected"}
    # Nothing ties these arguments to what the reviewer saw.
    refund = payments.refund(state["account_id"], state["amount"])
    return {"status": "submitted", "refund_id": refund.id}
```

Stage 2: the same node governed by VeriProof

Python

```python
from veriproof_sdk import GovernedActionContext, GovernReviewOptions, Veriproof

vp = Veriproof.from_env()
# At startup, register the app's current-eligibility check:
# reviews = await vp.reviews(store=vp.hosted_review_store(),
#                            revalidate=refund_still_eligible)

async def send_refund(context: GovernedActionContext) -> dict:
    # This review profile permits one correction: a lower refund amount.
    amount = context.input["amount"]  # Whole dollars in this example.
    if context.review_response is not None:
        match context.review_response["payloadReplacements"]:
            case [{"op": "replace", "path": "/amount", "value": value}]:
                if value["kind"] != "number":
                    raise ValueError("Expected a numeric refund amount")
                amount = value["value"]
            case _:
                raise ValueError("Expected one refund amount correction")
    if type(amount) is not int or not 0 < amount <= context.input["amount"]:
        raise ValueError("Refund exceeds the request or is invalid")
    refund = await payments.refund(context.input["account_id"], amount)
    return {"refund_id": refund.id}
async def issue_refund(state: RefundState) -> dict:
    result = await vp.govern(
        "refunds.issue",
        {"account_id": state["account_id"], "amount": state["amount"]},
        send_refund,
        idempotency_key=state["run_id"],
```

```
        facts={"refund_amount": state["amount"]},  # rules hold the threshold
        review=GovernReviewOptions(
            profile="refund-review",
            proposed_action_summary="Refund to the original payment method.",
        ),
    )
    if result.review is not None:
        # Escalate: store the checkpoint. After the signed decision arrives,
        # a worker calls reviews.resume(checkpoint, send_refund).
        await reviews.save(result.review)
        return {"status": "awaiting_review"}
    if not result.completed:
        return {"status": result.status.value, "reason": result.reason_code}
    return {"status": "submitted", **result.value}
```

Your application owns the callback endpoint and the resume worker. The SDK verifies each callback's Ed25519 signature against keys VeriProof publishes, checks that the decision belongs to this checkpoint, and lets one worker claim the continuation. The payment system reports whether the refund settles. The review profile allows a reviewer to lower the amount. If the reviewer changes $2,400 to $1,800, the handler reads the signed correction and sends $1,800 to the original account. Adapters apply the same wrapper without editing each tool. The LangGraph adapter governs a whole graph run or, through ToolNode's awrap_tool_call, each tool call. Other adapters cover the OpenAI Agents SDK, CrewAI, LlamaIndex, Pydantic AI, Semantic Kernel, Spring AI, and more, with SDKs for Python, TypeScript, .NET, Java, Go, and Rust.

3. Eliminating NIGO rework and the rubber-stamp trap

Structured in-flight resolution, and authority checked at the moment of decision.

A payment request can reach a reviewer because an amount is wrong or information is missing. A reviewer who can correct the amount can resolve the request without sending it back for re-entry.

Collect more than a yes-or-no answer

A review profile defines what the reviewer sees: the proposed action, the facts the application supplied, and the rule that asked for review. The reviewer can approve, reject with a written reason, or submit a response. A response can correct fields the profile marks as editable, such as lowering a refund from $2,400 to $1,800, or send the application a question. Every field is typed and validated.

The correction reaches the application in the signed callback. The application then runs the original action, runs it with the reviewer's response, or sends the work back for revision.

Verify identity and role authority at decision time

When a reviewer submits a decision, VeriProof checks the person’s access to the application, their assignment, and the permissions required for that case. The decision record identifies who acted.

Separation of duties can bar an application's owners from deciding its reviews. When the owner is the only eligible reviewer, they confirm explicitly before deciding, and the record notes it.

Review forms generated for each work app

The review profile defines which facts a reviewer sees and which fields they can correct. Connected Slack and Teams workflows bring the request to the reviewer. VeriProof checks authority and validates the submitted response.

Automated SLA timers and escalation chains

Review queues need owners and deadlines. VeriProof supports reminders and fallback routing so teams can follow up on overdue cases.

4. Cryptographic protocol and evidence schema

SHA-256 fingerprints, Merkle anchoring, signed exports, and independent verification.

Trace retention depends on the tool and its configuration. A trace helps an engineer investigate a run. A signed evidence pack lets an auditor check exported records against a trusted key and the ledger.

VeriProof records the policy decision, linked review activity, and outcome reported by the application. It commits record hashes to the tenant’s selected ledger in batches.

VeriProof evidence pack contents

The export scope determines which records accompany the integrity proof. The appendix shows an example of the signed manifest.

File	What it gives the reviewer
manifest.json	Package metadata: purpose, requester, record count, and whether payloads and redaction are included.
sessions/ and revisions/	The exported decision records. When revisions/ is present, the verifier hashes those exact bytes against the Merkle leaves.
review-records/	Review cases, continuation events, and the application's execution acknowledgments.
redaction-report.json	What was removed for this export and why, so reviewers can see how it differs from the source data.
chronology.pdf	A readable timeline of the included decisions for reviewers who don't work in JSON.

Independent verification

Auditors run veriproof-verify with the pack and a trusted signing key. It checks the signature, file inventory, and Merkle proofs locally. To confirm the anchor, it retrieves the finalized Solana transaction or checks Azure SQL Ledger against its protected digest storage.

A passing result confirms that the records covered by the pack match their ledger commitments.

Cross-framework regulatory mapping

For high-risk systems, the EU AI Act requires automatic event logging (Article 12) and effective human oversight, including the ability to override or stop the system (Article 14). Compliance officers map decision records to controls in the EU AI Act, ISO/IEC 42001, NIST AI RMF, GDPR, HIPAA, and SOC 2, and see which controls have current evidence. Gaps become remediation items with an owner and a due date.

5. Enterprise deployment topologies and fail-safe resilience

Where policy checks run, and what happens when VeriProof can't be reached.

A policy check sits in the request path, so where it runs and how it behaves under failure matter as much as the rules themselves.

Three deployment models for regulated enterprises

All three record decisions in the same format, so evidence packs and verification work the same way.

Deployment topology	Hosting and data residency	Evaluation path	Fail-safe behavior
1. Managed cloud control plane	VeriProof operates the hosted control plane on Azure.	The SDK requests a policy decision from the hosted service.	Without a valid decision, the protected action stays blocked.
2. Private Enterprise	Customer-hosted architecture in the organization's Azure subscription, with customer-managed access and ledger resources.	The application reaches the control plane through the customer's configured network.	Fails closed, as in the managed service.
3. VeriProof Edge Gateway	A container deployed near the application to evaluate eligible policies locally.	Signed policy bundles evaluated locally. Human review and shared controls stay central.	Keeps evaluating while its signed policy lease is valid, then fails closed until it reconnects. Evidence is journaled locally and delivered later.

Where the network hop matters, the Edge Gateway evaluates eligible rules beside the agent with no round trip to VeriProof.

6. Architectural trade-offs and total cost of ownership

When to build governance extensions in-house, and what keeps costing time after launch.

A capable platform team can build each of these pieces. The ongoing cost is keeping them consistent across every agent and team, and keeping the evidence acceptable to auditors as requirements change.

Control plane component	Internal build scope	Ongoing maintenance burden	VeriProof equivalent
Policy service with versioning and Shadow mode	Rule storage, an evaluation API, version history, and comparison of new rules against live traffic.	Rule migrations, rollout tooling, and change approval.	Included
Reviewer authority and signed callbacks	Permission checks at decision time, separation of duties, and callback signing with key rotation.	Identity provider changes, key rotation, and security fixes.	Included
Review forms and channels	Typed review forms, a reviewer inbox, and Slack and Teams apps that record decisions.	Keeping every channel in step with case state.	Included
Deadlines and escalation	Business calendars, reminders, fallback routing, and missed-deadline records.	Calendar, team, and org-chart changes.	Included
Evidence and verification	Append-only storage, hashing, ledger anchoring, signed exports, and a verifier auditors can run.	Format changes, proof recovery, and auditor questions.	Included

The public calculator and customer portal use the same planning assumptions: 40 developer hours to connect an application, then 120 hours a year to maintain its governance, compared with 320 under the current process. Track setup and ongoing work separately to see how closely your first integration matches the model.

7. Production rollout and verification checklist

Move from Shadow mode evaluation to enforced production control.

1. Instrument one high-impact boundary. Wrap it with the SDK or your framework's adapter, register a review profile and a callback URL, and confirm the round trip in a test environment.
2. Tune policies in Shadow mode. Shadow records what each rule would have done without changing behavior. Go through the would-be escalations and denials with the business owner and tune thresholds until the review load fits the team.
3. Enable enforcement and ledger-backed evidence. Move the rule to Enforced, staging the rollout when several applications share it. Then export an evidence pack and have someone outside the team run the verifier on it.

8. References

1. LangGraph, “Interrupts.” docs.langchain.com/oss/python/langgraph/interrupts
2. OpenAI, “Human-in-the-loop,” OpenAI Agents SDK. openai.github.io/openai-agents-python/human_in_the_loop
3. Temporal, “Approval pattern.” docs.temporal.io/design-patterns/approval
4. LangChain, “LangSmith administration overview: data retention.” docs.langchain.com/langsmith/administration-overview
5. Datadog, “Trace retention.” docs.datadoghq.com/tracing/trace_pipeline/trace_retention

6. MITRE, “CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition.” cwe.mitre.org/data/definitions/367.html
7. Regulation (EU) 2024/1689 (Artificial Intelligence Act), Articles 12 and 14. eur-lex.europa.eu/eli/reg/2024/1689/oj
8. ISO/IEC 42001:2023, Artificial intelligence management system. iso.org/standard/81230.html
9. NIST, “AI Risk Management Framework (AI RMF 1.0).” nist.gov/itl/ai-risk-management-framework
10. VeriProof, “How VeriProof checks evidence integrity.” veriproof.app/whitepapers/tamper-evidence
11. VeriProof, “Verify exported evidence.” docs.veriproof.app/agent-integrations/verification

Appendix: Inside an evidence pack

This excerpt from integrity-proof.json shows how the file inventory, signature, and ledger reference fit together.


```
{
  "contractVersion": 5,
  "canonicalizationVersion": "length-prefixed-binary-v5",
  "hashingVersion": "sha256-v1",
  "merkleVersion": "sha256-hex-leaf-v1",
  "packageId": "pkg-4f1c09ab-20261024",
  "entries": [
    { "path": "sessions/refund-7c41.json", "length": 3968, "sha256": "3ce25a3d…" }
  ],
  "packageCommitmentHash": "77250bd7f85b8a502ab6f8888f6caf09011b4d477069a1fe…",
  "signature": { "algorithm": "ecdsa-p256-sha256-v1", "value": "MEYCIQCOPGZx1X…" },
  "proofs": [{
      "sessionId": "refund-7c41",
      "inclusion": {
        "root": "a86de7d878ecc28d7f07d6645eeef5ab102e913a3571f1b4…",
        "leafIndex": 2,
        "siblings": [{ "hash": "8e1b57407a84c29e…", "side": "Right" }]
      },
      "anchor": {
        "provider": "azure_sql_ledger",
        "anchoredAt": "2026-10-24T14:22:09Z",
        "azureSqlLedger": { "database": "vp-ledger", "anchorBatchId": "5b0e2f61-…" }
      }
    }
  ]
}
```


Try one workflow with your team

Protect one test tool call. Pause a refund, lower the amount during review, and resume it. Check that the payment service received the corrected amount. Then verify the exported record.

Start your free 60-day trial

veriproof.app/start