Nick Clark · Published April 25, 2026 · PDF

How to Load an Agent Skill or Adapter Only After It Has Been Earned

You want an agent to gain a skill or load an adapter module only once it has actually demonstrated competence, not because a config flag or a role assignment said it could. This guide walks through an architecture for evidence-gated, runtime skill loading, where a scoped module is admitted only after the governing competency and authorization conditions both hold. The approach described here is disclosed in United States Patent Application 19/647,395, under the LLM and Skill Gating inventive step. It is an architecture you build, not a library you install.

What You Are Building

You are building a gate that stands between an agent and a capability the agent wants to use, and that opens only when the agent has accumulated real evidence of competence in that specific capability. The skill or adapter module that implements the capability does not load at startup and does not load because a permission flag is set. It loads at runtime, only after the gate opens, and it can be withdrawn again if the evidence of competence decays.

This is the problem a growing class of builders now faces: an agent framework where every agent can, in principle, invoke every tool or load every adapter, and where the only thing standing between an underqualified agent and a high-consequence action is

a boolean in a config file. If you are shipping agents that touch money, infrastructure, medical workflows, or physical equipment, you want the loaded surface area of each agent to reflect what it has demonstrably earned. That is what this architecture gives you a way to express.

The design here is disclosed in United States Patent Application 19/647,395. This guide teaches the architecture. You implement it in your own stack.

Why the Obvious Approaches Fall Short

The usual way to control what an agent can do is role-based or attribute-based access control. You assign the agent a role, the role carries a permission set, and a check at call time confirms the permission is present. This is a sound, well-understood pattern for most software. Its limitation, for this problem, is that a role attests to an assignment, not to demonstrated ability. A role says someone decided this agent should be allowed to do the thing. It does not say the agent can do the thing well, and it does not notice when the agent stops being able to.

Feature flags and capability manifests have the same shape. They are declarations set ahead of time by a human or a deployment pipeline. They are static: once flipped on, they stay on until someone flips them off. Nothing in the mechanism observes the agent's actual performance and reconsiders.

Credential systems, including signed tokens and certificates, are stronger because they are verifiable and can carry an expiration. But a conventional credential still attests to a past event: a training completion, an issuance decision, a role grant at a moment in time. It asserts identity or entitlement at discrete points and does not, on its own, establish a continuous link to whether the holder currently performs competently.

The structural gap in all three is the same. They separate the grant of a capability from any ongoing measurement of competence at that capability. The filed disclosure frames its capability gate specifically against this: it grants access based on accumulated performance evidence rather than on credentials, roles, or static permission assignments, and it treats the grant as a continuous evaluation rather than a one-time assessment.

The Architecture

The disclosure organizes this into a small number of cooperating parts. Each is described below and traced to the filing.

The capability gate. The gate is a governed evaluation point that sits between a requester (which the disclosure allows to be a human operator, a semantic agent, or a composite system) and a capability the requester wants to exercise. It evaluates the requester's accumulated evidence of competence in the relevant domain against defined competency thresholds and produces one of two outcomes: a progressive unlock that grants access, or a regression/revocation that denies or withdraws access based on evidence that competence has degraded below the required threshold. Crucially, the disclosure describes the gate as a continuous evaluation: because performance is monitored after a capability is granted, the gate may later close and revoke access it previously opened.

The curriculum engine and progressive unlock. The evidence the gate consumes is produced by a curriculum engine. In the disclosure, this engine defines, for each gated capability, a set of learning objectives, a set of assessment instruments, a sequencing policy governing the order in which they are presented, and a mastery threshold per objective. It implements progressive unlock: a capability is not granted in one assessment event but is unlocked in stages, exposing the requester to simpler aspects before granting the more complex or higher-risk aspects, so that the accumulated evidence reflects competence across the full scope of the capability rather

than a single passing score. The disclosure also makes each curriculum a governed object: its definition, sequencing, and modification are subject to policy constraints and recorded in the curriculum's lineage, so a curriculum cannot be quietly weakened, shortened, or bypassed without an attributable, auditable policy change.

The certification token. When the gate opens, the disclosure has the system generate a certification token: a cryptographically signed data object attesting to the holder's demonstrated mastery of a specific capability, at a specific time, under specific assessment conditions. The disclosure is explicit that this is not a conventional credential, not a role assignment, permission grant, or static badge, but a time-bounded, evidence-backed attestation subject to expiration, revocation, and revalidation. The token's described fields include a capability identifier, the holder's identity, an evidence hash (a cryptographic hash of the evaluated evidence corpus, letting a verifier confirm the token was issued against specific evidence without seeing the evidence), an issuance timestamp, an expiration timestamp, the policy scope under which it was issued, the issuing authority, a device entropy binding to the device from which the mastery evidence was submitted, and the issuing authority's signature.

The token lifecycle and the deployment gate. The disclosure gives the token a defined lifecycle: active on issuance (presentable to gates as evidence of mastery), inactive on expiration (the holder must re-demonstrate mastery), invalidated on revocation (triggered by evidence of mastery regression, incident reports, or governance intervention, regardless of expiration), and revalidated when a re-assessment issues a fresh token with new evidence bindings. Each transition is recorded as a governed event in the holder's lineage. For cross-platform use, a receiving system's deployment gate verifies the token's signature against the issuing authority's public key, checks its expiration, and evaluates its policy scope for compatibility with the receiving system's own governance requirements before accepting it, and may still impose its own capability gate on top.

The competency-and-authorization intersection. This is the part that makes the loaded surface truthful. The disclosure keeps capability determination and permission determination in architecturally separate subsystems with no bidirectional dependency: the capability subsystem does not consult governance policy, and the governance subsystem does not consult capability. Their independent determinations are combined at an execution synthesis gate where both conditions, capability and authorization, must be satisfied for execution to proceed. A skill loads only where earned competence and current authorization intersect. Being highly authorized does not manufacture competence, and being highly competent does not manufacture permission.

How to Approach the Build

The steps below are an implementation path faithful to the disclosed architecture. The interface sketches are illustrative only and are not a package you can install.

1. **Enumerate capabilities as scoped modules.** Split the surface area you want to gate into discrete skills or adapters, each behind its own identifier. The unit of gating is a capability, so a skill should be independently loadable and independently deniable.
2. **Author a curriculum per capability.** For each capability, define its learning objectives, the assessment instruments that measure them, a sequencing policy, and a mastery threshold per objective. Treat the curriculum as a governed, versioned object with a lineage, so no one can silently lower a threshold.
3. **Accumulate evidence, do not just record a pass.** Feed assessment outcomes and continuous operational monitoring into an evidence store keyed to the requester and the capability. The gate reads accumulated evidence, so a single passing event is not the same as a standing grant.
4. **Implement the gate as a continuous evaluator.** The gate compares accumulated evidence against thresholds and returns unlock or regression/revocation. Re-run it on new evidence, not only at first request, so

degradation can close a gate that was previously open.

```
# illustrative only, faithful to the disclosed roles
decision = capability_gate.evaluate(
    requester, capability_id, evidence_store.for(requester, capability_id
if decision.unlocked:
    token = issue_certification_token(requester, capability_id, evidence_h
```

5. **Issue the certification token on unlock.** Populate the disclosed fields: capability id, holder identity, evidence hash, issuance and expiration timestamps, policy scope, issuing authority, device entropy binding, and signature. Record the issuance in the holder's lineage.
6. **Gate the runtime load on the intersection.** Keep competency and authorization in separate subsystems. At the point where the agent would load the scoped module, require both a currently valid token (verified signature, unexpired, policy scope compatible) and an independent authorization decision. Load only where both hold.
7. **Wire the lifecycle back to loading.** On expiration or revocation, the token stops being valid evidence; the load path must consult live token state, not a cached grant, so a revoked skill actually unloads or fails closed.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and nothing here that just works out of the box. You implement the gate, the curriculum engine, the evidence store, the token format, and the intersection check in your own stack, and you make the real engineering decisions the disclosure leaves to the implementer: what counts as evidence, where thresholds sit, how tokens are signed and verified, and how often the gate re-evaluates.

It is a disclosed approach in a patent filing, not a benchmarked or production-proven system. This guide reports no performance numbers, latency figures, or accuracy guarantees, because the filing is an architectural disclosure and asserting such numbers would be inventing them. The security of your build rests on choices outside the architecture: the quality of your evidence, the integrity of your signing keys, and the soundness of your verification.

The pattern also does not fit every problem. If your capabilities are low-consequence, or if a static role grant already matches your risk, the machinery here is overhead you do not need. Its value shows up when the cost of an underqualified or degraded agent acting is high enough to justify tying the loaded surface to standing, revocable evidence of competence.

Disclosure Scope

The architecture described in this guide, including evidence-based capability gating, the curriculum engine and progressive unlock, the certification token and its lifecycle, and the separation of competency from authorization at an execution synthesis gate, is disclosed in United States Patent Application 19/647,395. This guide is educational. It explains an approach a developer can build and is not a warranty, a specification of a shipping product, or an offer of software. Any implementation, and the security and correctness of that implementation, is the responsibility of the reader.

LLM & Skill Gating (/llm-skill-gating)

All 40 steps → (/inventive-steps)

The model proposes. The agent decides.

Chapter 7 (/patents/19-647395/chapters/llm-skill-gating)

PRIMARY TECHNICAL DISCLOSURE

- AI-Mediated Curriculum and Progressive Capability Unlocking Using Semantic Performance States (/articles/ai-mediated-curriculum-and-progressive-capability-unlocking-using-semantic-performance-states)

SECONDARY TECHNICAL

- LLM as Structurally Untrusted Proposal Generator (/articles/llm-skill-gating/untrusted-proposals)
- Mutation-Validation-Arbitration Pipeline (/articles/llm-skill-gating/mutation-validation-pipeline)
- Hallucination Prevention via Structural Starvation (/articles/llm-skill-gating/structural-starvation)
- Trust Weight Calibration and Decay (/articles/llm-skill-gating/trust-weight-calibration)
- Evidence-Based Capability Gating (/articles/llm-skill-gating/evidence-gating)
- Certification Token Generation (/articles/llm-skill-gating/certification-tokens)
- Narrative State and Personality Architecture (/articles/llm-skill-gating/narrative-personality)
- Skill Regression Detection and Capability Revocation (/articles/llm-skill-gating/regression-detection)
- Arbitration as Semantic Event (/articles/llm-skill-gating/arbitration-events)
- Structural Starvation as a Composable Safety Primitive (/articles/llm-skill-gating/starvation-composability)
- Multi-Turn Memory Isolation (/articles/llm-skill-gating/memory-isolation)
- Curriculum Engine Progressive Unlock (/articles/llm-skill-gating/curriculum-engine)
- Multimodal Evaluation Pipeline (/articles/llm-skill-gating/multimodal-evaluation)
- Multimodal Anti-Gaming Substrate (/articles/llm-skill-gating/anti-gaming)
- Professional Skill Gating Applications (/articles/llm-skill-gating/professional-gating)
- Embodied Skill Gating (/articles/llm-skill-gating/embodied-gating)
- Biological Identity Skill Binding (/articles/llm-skill-gating/biological-binding)
- Security and Drift Detection Layer (/articles/llm-skill-gating/security-layer)
- Validation Feedback Asymmetry (/articles/llm-skill-gating/feedback-asymmetry)

APPLICATIONS · GENERAL

- Progressive AI Agent Deployment: Granting Authority Through Earned, Continuously-Evidenced Capability (/articles/llm-skill-gating/enterprise-progressive-deployment)
- Educational Platform Competency Through Structural Certification (/articles/llm-skill-gating/educational-competency)

- How to License Medical AI: Evidence-Gated Clinical Capability and Competence Governance (/articles/llm-skill-gating/medical-licensing)
- Jurisdiction-Gated Legal AI: Certifying Practice-Area Competence Before an LLM Gives Advice (/articles/llm-skill-gating/legal-practice-certification)
- AI Flight Training That Gates Pilot Privileges on Demonstrated Competence, Not Credentials (/articles/llm-skill-gating/aviation-pilot-training)
- AI Financial Advisor Certification: Fiduciary-Grade Skill Gating for Investment Advice (/articles/llm-skill-gating/financial-advisor-certification)
- Skill Gating for Cybersecurity AI: Earning Dangerous Capabilities Through Evidence (/articles/llm-skill-gating/cybersecurity-skill-progression)
- LLM and Skill Gating for Manufacturing Quality Systems (/articles/llm-skill-gating/manufacturing-quality)
- Decentralized Agent Skill Marketplace: Cryptographic Skill-to-Authority Binding for AI Agent Platforms (/articles/llm-skill-gating/agent-skill-marketplace)
- Runtime LoRA Adapter Admission Control for Regulated AI Deployment (/articles/llm-skill-gating/runtime-lora-loading)
- Multi-Authority AI Licensing Compliance at the Inference Boundary (/articles/llm-skill-gating/composite-licensing-intersection)

APPLICATIONS · SPECIFIC

- Duolingo Alternative: Evidence-Gated Capability vs Content Unlock (/articles/llm-skill-gating/duolingo)
- Khan Academy Khanmigo vs Evidence-Gated Capability Unlock (/articles/llm-skill-gating/khan-academy)
- Coursera Alternative for Competence-Verified Credentials: Governed Skill Gating (/articles/llm-skill-gating/coursera)
- GitHub Copilot vs Evidence-Gated Code Generation (/articles/llm-skill-gating/github-copilot)
- Pearson Alternative for Governed Capability Progression: Evidence-Gated Skill Unlocking (/articles/llm-skill-gating/pearson)
- Chegg vs Evidence-Gated Capability Unlock: A Governed Alternative to Answer Access (/articles/llm-skill-gating/chegg)
- Grammarly Alternative for Evidence-Gated Writing Capability (/articles/llm-skill-gating/grammarly)
- Is there a Photomath alternative that makes students earn each solution? (/articles/llm-skill-gating/photomath)
- Century Tech Alternative: Evidence-Gated Mastery Beyond Adaptive Recommendation (/articles/llm-skill-gating/century-tech)

- Squirrel AI vs evidence-based capability gating: which certifies mastery? (/articles/llm-skill-gating/squirrel-ai)
- Anthropic Skills Alternative: Consumer-Side Certification for Governed Skill Admission (/articles/llm-skill-gating/anthropic-skills)
- OpenAI Custom Actions Alternative: Evidence-Gated Action Authority (/articles/llm-skill-gating/openai-custom-actions)
- Google Gemini Extensions vs Evidence-Gated Capability Unlocking (/articles/llm-skill-gating/google-gemini-extensions)
- Microsoft Copilot Studio Alternative for Sovereign and Air-Gapped Agent Governance (/articles/llm-skill-gating/microsoft-copilot-studio)
- HuggingFace PEFT vs Evidence-Gated Capability: Governed Skill Activation Beyond Adapter Loading (/articles/llm-skill-gating/huggingface-peft)
- Meta Llama Guard vs Governed Skill Gating: Content Filtering Beyond the Safety Classifier (/articles/llm-skill-gating/meta-llama-llama-guard)
- OpenAI Operator vs Evidence-Gated Agent Execution (/articles/llm-skill-gating/openai-gpt4o-operator)
- Windsurf Alternative: Evidence-Gated Agent Capability Beyond Workspace Toggles (/articles/llm-skill-gating/codeium-windsurf)

LLM & Skill Gating overview → (/llm-skill-gating)