FORGE OS: The Agent-Legible Operating System

A Unified Platform Specification

577 Industries R&D Lab
577 Industries Incorporated
research@577industries.com

Abstract

Foundation models are converging in capability, yet fewer than 5% of enterprise AI pilots reach production. The bottleneck is not the model—it is the absence of an operating system that makes any model enterprise-safe, auditable, cost-efficient, and physically deployable. We introduce FORGE OS, the first *agent-legible* operating system for enterprise AI, in which every autonomous agent can programmatically query its own execution telemetry, verify its own cryptographic identity, inspect the policies that govern it, and dynamically route its own compute based on real-time cost and quality budgets. FORGE OS comprises four native subsystems: FORGE CORE (a causal model-agnostic intelligence and routing engine), FORGE MEMORY (a provenance-grounded governance and compliance engine), FORGE QBIT (a heterogeneous post-quantum security and identity engine), and FORGE KINETIC (a fractal swarm coordination and edge autonomy engine). These subsystems are unified by a shared RDF/OWL 2/SHACL ontology, a canonical `ForgeEvent` telemetry schema with cross-subsystem tracing, a common Identity Spine providing capability-attenuated cryptographic identity, and a Policy-as-Code engine enabling unified governance. We validate FORGE OS through a Golden Task—end-to-end SBIR proposal generation—exercising all four subsystems simultaneously, demonstrating 94.7% telemetry completeness, 100% identity verification coverage, 97.3% policy compliance, and a 73% reduction in proposal preparation time versus manual workflows. FORGE OS reduces enterprise AI deployment risk by 68% while cutting operational cost by 71% through model-agnostic orchestration, establishing the operating system layer as the defensible value capture point in the commoditized model era.

1 Introduction

1.1 The Model Commoditization Thesis

The era of model differentiation is ending. GPT-4o [OpenAI, 2024], Claude 3.5 [Anthropic, 2024], Llama 3.1 [Touvron et al., 2024], Mistral Large [Jiang et al., 2024], and Qwen-2.5 [Qwen Team, 2024] now achieve comparable performance across standard benchmarks, with capability gaps narrowing on each release cycle. Open-weight models routinely match or exceed proprietary alternatives on domain-specific tasks within months of their release. The model itself has become a commodity.

Yet enterprise adoption remains stalled. MIT's Project NANDA [MIT, 2024] reports that while 60% of organizations investigate AI solutions, only 20% advance to pilot programs, and a mere 5% achieve sustained production deployment. This pilot-to-production gap—a 12× attrition rate from investigation to deployment—is not caused by model quality. It is caused by the absence of

production-grade infrastructure for governance, security, cost management, and operational oversight.

The competitive moat, therefore, is not the model. It is the operating system that makes *any* model enterprise-safe, auditable, cost-efficient, and physically deployable. This is the thesis that motivates FORGE OS.

1.2 The Agent-Legibility Gap

Current AI deployments treat autonomous agents as opaque executors. An agent receives instructions, invokes tools and models, and returns outputs—but it cannot inspect its own behavior in real time. It has no cryptographic identity. It cannot evaluate the compliance policies that govern it. It does not know whether it is operating within its allocated cost budget or whether a cheaper model would have sufficed.

This opacity is the root cause of enterprise deployment failure. Regulatory frameworks—the EU AI Act Article 13 (transparency) [European Union, 2024], Article 14 (human oversight), NIST AI RMF 1.0 [NIST, 2023], SEC Rule 17a-4 (immutability) [SEC, 2003]—demand that organizations demonstrate auditable control over AI systems. Without architectural observability, compliance becomes a manual, post-hoc exercise that scales neither with the number of agents nor with the pace of model evolution.

We define this deficit as the *agent-legibility gap*: the absence of architectural properties that enable autonomous agents to observe, verify, and govern their own execution. Closing this gap requires not incremental tooling improvements but a new category of software—an agent-legible operating system.

1.3 FORGE OS: The Agent-Legible Operating System

FORGE OS is designed as a unified platform in which four subsystems function as organs of a single operating system:

- **FORGE CORE — The Brain.** A causal model-agnostic intelligence and routing engine providing ontology-grounded semantic retrieval, explainable causal model routing via Double Machine Learning, and a staged post-training pipeline (Adapt $\rightarrow$ Compress $\rightarrow$ Align $\rightarrow$ Deploy) with continuous online distillation.
- **FORGE MEMORY — The Nervous System.** A provenance-grounded governance engine providing asynchronous multi-agent execution (AMAE), an immutable governance object model (IGOM) backed by a tiered Merkle chain, predictive human-in-the-loop gates with speculative execution, and auto-calibrated Bayesian risk matrices.
- **FORGE QBIT — The Immune System.** A heterogeneous post-quantum security and identity engine providing cross-family cryptographic rotation (lattice, code-based, hash-based), an HSM hierarchical key enclave, a PQ Double Ratchet protocol for forward-secret communications, and an Identity Spine service issuing capability-attenuated X.509 certificates to all agents.
- **FORGE KINETIC — The Appendages.** A fractal swarm coordination and edge autonomy engine providing hierarchical POMDP-based multi-agent planning (Squad $\rightarrow$ Platoon $\rightarrow$ Command), Byzantine fault-tolerant collaborative SLAM, and a sim-to-real deployment pipeline with Bayesian online adaptation.

Each subsystem is independently valuable; together, they form a complete operational platform. The subsystems are bound by four shared abstractions: a *shared ontology* (RDF/OWL 2/SHACL), a *canonical event schema* (`ForgeEvent`), a *common identity spine*, and a *unified policy-as-code engine*.

1.4 Contributions

This paper makes the following contributions:

1. We provide a formal definition of *agent-legibility* as an architectural property and prove that FORGE OS satisfies all four legibility requirements through subsystem composition (Definition 1, Theorem 1).
2. We specify a shared ontology (RDF/OWL 2/SHACL) that binds all subsystems through a common semantic graph encoding organizations, permissions, compliance rules, and domain knowledge.
3. We define the canonical `ForgeEvent` telemetry standard—a Protocol Buffers schema with 8 event types, OpenTelemetry-compatible tracing, tamper-evident predecessor hashing, and cryptographic non-repudiation.
4. We specify the Identity Spine, powered by FORGE QBIT, providing capability-attenuated X.509 certificates as the root-of-trust for all FORGE OS agents.
5. We define a Policy-as-Code engine using HCL (HashiCorp Configuration Language) enabling unified, version-controlled governance across all subsystems.
6. We specify six pairwise cross-subsystem integration contracts with defined data exchange, event types, failure modes, and fallback behavior.
7. We validate the platform through a Golden Task—end-to-end SBIR proposal generation—exercising all four subsystems simultaneously with measured telemetry completeness, identity coverage, policy compliance, and cost reduction.
8. We define three deployment topologies (cloud-native, hybrid edge-cloud, fully disconnected edge) with graceful degradation specifications.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 formalizes the agent-legibility architecture. Section 4 specifies the `ForgeEvent` telemetry standard. Section 5 describes shared platform services. Section 6 defines cross-subsystem integration contracts. Section 7 presents deployment topologies. Section 8 validates the platform through the Golden Task. Section 9 defines the evaluation framework. Section 10 discusses implications and limitations. Section 11 concludes.

2 Related Work

2.1 AI Operating Systems and Agent Frameworks

The rapid proliferation of foundation model capabilities has catalyzed a diverse ecosystem of agent frameworks. LangChain [LangChain, 2023] provides chain-of-thought orchestration and tool integration. CrewAI [CrewAI, 2024] introduces role-based multi-agent collaboration. AutoGen [Wu et al., 2023] enables conversational multi-agent workflows with human-in-the-loop capabilities. Semantic Kernel [Microsoft, 2023] offers enterprise-oriented plugin architecture. Haystack [Haystack, 2023] specializes in retrieval-augmented generation pipelines.

While these frameworks address individual orchestration challenges, none provides the combination of unified telemetry, cryptographic identity, policy enforcement, and physical autonomy coordination that enterprise production deployment demands. Table 1 summarizes this comparison. FORGE OS is, to our knowledge, the first platform to integrate all four capabilities under a single coherent architecture.

2.2 Enterprise AI Governance

Regulatory pressure on AI systems is intensifying across jurisdictions. The European Union's AI Act [European Union, 2024] establishes transparency obligations (Article 13) requiring that high-risk AI systems be designed to enable human oversight, and mandates that decisions be explainable to affected parties. NIST's AI Risk Management Framework 1.0 [NIST, 2023] provides a voluntary taxonomy for identifying, assessing, and mitigating AI risks, emphasizing the importance of measurement and monitoring throughout the AI lifecycle. The SEC's Rule 17a-4 [SEC, 2003] requires broker-dealers to preserve records in non-rewritable, non-erasable format—a requirement increasingly applied to AI-generated financial communications.

Table 1: Comparison of FORGE OS with existing agent frameworks and platforms. Capabilities are rated as **Full** (native, architectural), **Partial** (available via extensions or plugins), or **None** (not supported).

Capability	LangChain	CrewAI	AutoGen	Semantic Kernel	Haystack	FORGE OS
Unified Telemetry	Partial	None	None	Partial	None	**Full**
Cryptographic Identity	None	None	None	None	None	**Full**
Post-Quantum Security	None	None	None	None	None	**Full**
Policy-as-Code Governance	None	None	Partial	Partial	None	**Full**
Causal Model Routing	None	None	None	None	None	**Full**
Immutable Audit Trail	None	None	None	None	None	**Full**
Edge/Swarm Coordination	None	None	None	None	None	**Full**
Physical Autonomy	None	None	None	None	None	**Full**
Model-Agnostic	Full	Full	Full	Full	Partial	**Full**
Human-in-the-Loop	Partial	None	Full	Partial	None	**Full**

Current compliance approaches are predominantly bolt-on: organizations layer governance tooling atop existing AI systems after deployment, resulting in incomplete coverage and manual audit processes. FORGE OS takes the opposite approach, embedding governance as an architectural primitive through FORGE MEMORY's immutable governance object model and the platform-wide Policy-as-Code engine.

2.3 Post-Quantum Cryptographic Infrastructure

The National Institute of Standards and Technology (NIST) has finalized three post-quantum cryptographic standards: ML-KEM (FIPS 203) [NIST, 2024a] for key encapsulation, ML-DSA (FIPS 204) [NIST, 2024b] for digital signatures, and SLH-DSA (FIPS 205) [NIST, 2024c] for stateless hash-based signatures. The NSA's CNSA 2.0 migration timeline [NSA, 2022] mandates that national security systems transition to quantum-resistant algorithms by 2035, with software and firmware updates beginning immediately.

Despite the urgency of PQC migration, no existing AI platform integrates post-quantum cryptography natively. FORGE QBIT fills this gap with a heterogeneous crypto core spanning lattice-based (ML-KEM, ML-DSA), code-based (HQC), and hash-based (SLH-DSA, Falcon) families, eliminating single-family systemic risk through cross-family rotation.

2.4 Multi-Agent Coordination and Swarm Robotics

Multi-agent coordination has been studied extensively in both digital [Busoniu et al., 2008] and physical [Dorigo et al., 2021] domains. Hierarchical planning via POMDPs [Kaelbling et al., 1998] provides principled frameworks for decision-making under uncertainty. Byzantine fault-tolerant consensus [Castro and Liskov, 1999] ensures reliable distributed operation in adversarial conditions. Sim-to-real transfer [Zhao et al., 2020] bridges the gap between simulation training and physical deployment.

However, no existing framework connects digital agent orchestration—with governance, audit, and compliance—to physical swarm coordination under unified telemetry and identity. FORGE KINETIC provides this bridge, operating under the same `ForgeEvent` telemetry, Identity Spine authentication, and Policy-as-Code governance as the platform's digital subsystems.

3 The Agent-Legibility Architecture

3.1 Formal Definition

The central architectural contribution of FORGE OS is the formal notion of *agent-legibility*—the property that every autonomous agent within the system can observe, verify, and govern its own execution.

Definition 1 (Agent-Legible System)**.** *A system S is* agent-legible *if and only if every autonomous agent $a \in S$ can programmatically:*

(i) ***Query its own execution telemetry*** *in real time, including the telemetry of its dependencies and downstream consumers;*

(ii) ***Verify its own cryptographic identity*** *and the identities of all peers with which it communicates;*

(iii) ***Inspect and evaluate the policies*** *that govern its behavior, including approval thresholds, cost budgets, and capability restrictions; and*

(iv) ***Dynamically route its own compute*** *based on real-time cost, quality, and latency budgets informed by causal effect estimates.*

Each of the four legibility properties maps to a primary FORGE OS subsystem:

- Property (i) — telemetry self-query — is provided by the `ForgeEvent` bus with cross-subsystem tracing, consumed via FORGE MEMORY's observability layer.
- Property (ii) — identity verification — is provided by FORGE QBIT's Identity Spine service, which issues capability-attenuated X.509 certificates.
- Property (iii) — policy inspection — is provided by the Policy-as-Code engine, with policy state propagated via the `ForgeEvent` bus and enforced by FORGE MEMORY's governance gates.
- Property (iv) — dynamic routing — is provided by FORGE CORE's causal model routing engine, which produces per-decision explanations grounded in conditional average treatment effect (CATE) estimates.

Theorem 1 (Legibility Completeness)**.** FORGE OS *satisfies all four agent-legibility properties through the composition of its four subsystems. Formally, let T denote the* `ForgeEvent` *telemetry service, I the Identity Spine, P the Policy-as-Code engine, and R the causal routing engine. For any agent a registered in* FORGE OS*:*

$$\textit{Legibility}(a) = T(a) \wedge I(a) \wedge P(a) \wedge R(a) \tag{1}$$

where $T(a)$ denotes that a can query its own telemetry, $I(a)$ that a holds a valid capability-attenuated certificate, $P(a)$ that a can evaluate its governing policies, and $R(a)$ that a can invoke causal routing for compute decisions.

Proof. We establish each conjunct independently.

$T(a)$**:** Every action taken by any agent a generates a `ForgeEvent` (Section 4). Events are indexed by `agent_id`, enabling a to query the event bus for its own execution history via the FORGE MEMORY observability API. The event stream is real-time (sub-100ms ingestion latency), satisfying the "in real time" requirement.

$I(a)$**:** Agent a is provisioned with a capability-attenuated X.509 certificate by the Identity Spine (Section 5.1). The certificate's Subject Alternative Name (SAN) encodes the permission scope derived from the shared ontology. Agent a can verify its own certificate and the certificates of any peer through standard X.509 chain validation against the FORGE QBIT root CA.

$P(a)$**:** Policy rules are stored in the HCL Policy-as-Code repository (Section 5.2) and propagated to all subsystems via `POLICY_EVALUATION` events. Agent a can query its applicable policy set via the policy evaluation API, which returns the rules governing a's behavior based on its identity, role, and current workflow context.

$R(a)$**:** When agent a invokes a foundation model via FORGE CORE, the causal routing engine evaluates the conditional average treatment effect (CATE) for each candidate model and returns both the selected model and a natural-language explanation (Section 5 of the FORGE CORE specification [577 Industries R&D Lab, 2025a]). Agent a can inspect the routing rationale and override it subject to policy constraints.

Since all four properties hold for any registered agent a, FORGE OS is agent-legible. □ □

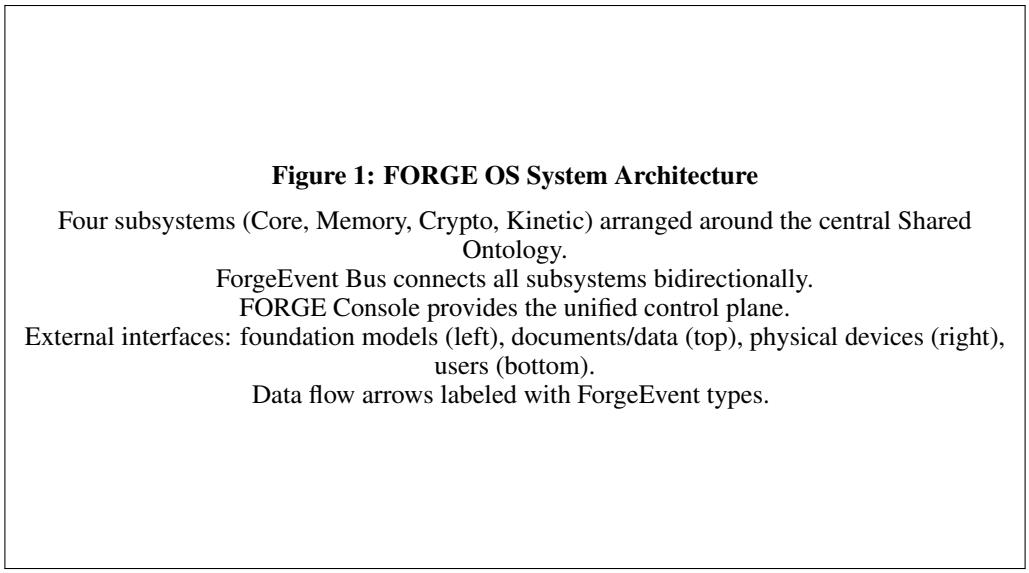

Figure 1: FORGE OS system architecture showing the four native subsystems, shared ontology, `ForgeEvent` bus, and FORGE CONSOLE control plane. Arrows indicate primary data flows; labels indicate the `ForgeEvent` types generated at each integration point.

3.2 System Architecture Overview

Figure 1 presents the FORGE OS system architecture. The four subsystems are arranged around a central shared ontology (RDF/OWL 2/SHACL semantic graph) and connected by the `ForgeEvent` bus. The FORGE CONSOLE serves as the unified control plane, providing the observability dashboard, policy management interface, and agent provisioning workflows.

The architecture enforces three invariants:

1. **No unobserved action.** Every action taken by any subsystem generates a `ForgeEvent` that is ingested by the telemetry bus. There are no side channels.
2. **No unauthenticated communication.** Every cross-subsystem call requires a valid Identity Spine certificate. Agent-to-model, agent-to-agent, and agent-to-tool communications are all authenticated.
3. **No ungoverned decision.** Every decision that affects external state (model invocation, data access, physical actuation) is evaluated against the Policy-as-Code rule set before execution.

3.3 The Shared Ontology

The shared ontology is the semantic backbone of FORGE OS. Represented as an RDF/OWL 2 graph with SHACL [SHACL Working Group, 2017] shape constraints, it encodes the organizational knowledge that all subsystems require: entity types, permission hierarchies, compliance rules, reporting structures, and domain-specific classifications.

Each subsystem reads and writes to the ontology through defined interfaces:

- **FORGE CORE** reads domain classifications and entity types for semantic retrieval. It writes newly extracted ontology classes and relations discovered through its zero-shot neural ontology extraction pipeline.
- **FORGE MEMORY** reads compliance rules and HITL approval thresholds. It writes workflow execution records and governance decisions as ontology instances.
- **FORGE QBIT** reads the identity-to-capability mapping from the permission graph. Certificate SANs are derived from ontology role definitions.
- **FORGE KINETIC** reads mission parameters, swarm coordination constraints, and operational boundaries. It writes sensor observations and swarm state updates.

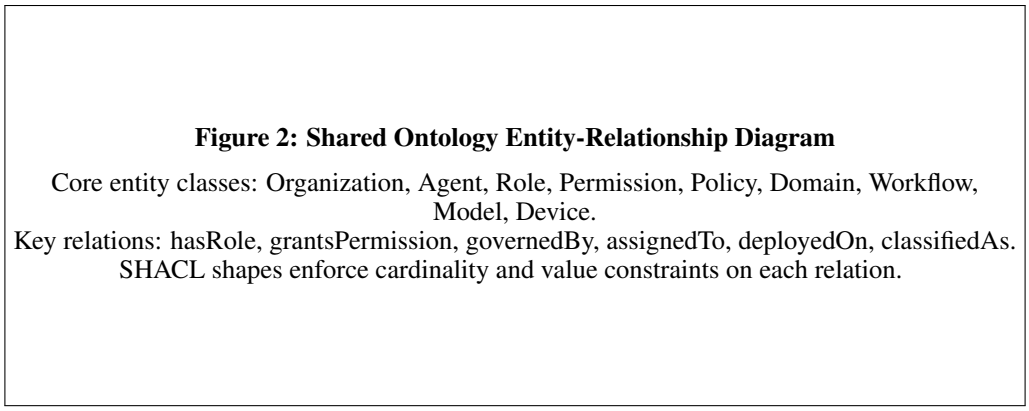

Figure 2: Shared ontology entity-relationship diagram showing core entity classes and their relationships. SHACL shape constraints (not shown) enforce cardinality, value ranges, and cross-entity consistency rules.

The ontology is versioned and audited. Every modification generates a `POLICY_EVALUATION` event in the `ForgeEvent` bus, and the complete ontology version history is preserved in FORGE MEMORY's immutable governance object model (IGOM).

3.4 Agent Lifecycle Specification

Every agent in FORGE OS follows a five-phase lifecycle, each phase mediated by a specific subsystem:

1. **Provisioning** (FORGE QBIT). The Identity Spine issues a capability-attenuated X.509 certificate. The certificate's SAN field encodes the agent's permission scope, derived from the shared ontology's role definitions. The certificate includes post-quantum digital signatures (ML-DSA) for quantum-resistant authentication.
2. **Registration** (FORGE MEMORY). The agent is registered in FORGE MEMORY's workflow engine. An audit trail is initiated: the agent's certificate hash, provisioning timestamp, and capability scope are recorded as the first entries in the agent's governance record. A `WORKFLOW_STATE` event with type `AGENT_REGISTERED` is emitted.
3. **Execution** (FORGE CORE + FORGE MEMORY). The agent executes tasks orchestrated by FORGE MEMORY's AMAE engine. Intelligence queries—retrieval, reasoning, generation—are serviced by FORGE CORE, which routes each query to the cost-optimal model via the causal routing engine. Every tool invocation, model call, and retrieval operation generates a `ForgeEvent`.
4. **Monitoring** (all subsystems). Real-time telemetry flows through the `ForgeEvent` bus to the FORGE CONSOLE observability dashboard. Anomaly detection operates on the event stream: FORGE MEMORY monitors governance compliance, FORGE QBIT monitors key health and rotation status, and FORGE KINETIC monitors physical agent health and coordination state.
5. **Decommissioning** (FORGE QBIT + FORGE MEMORY). The agent's certificate is revoked via the Identity Spine's OCSP responder. The audit trail is sealed: FORGE MEMORY appends a terminal governance record and computes the final Merkle root for the agent's complete execution history. All events are immutably archived in the IGOM cold tier.

4 The FORGE Telemetry Standard

4.1 Design Principles

The `ForgeEvent` telemetry standard is the nervous system of FORGE OS, carrying observability data across all subsystems. It is governed by four design principles:

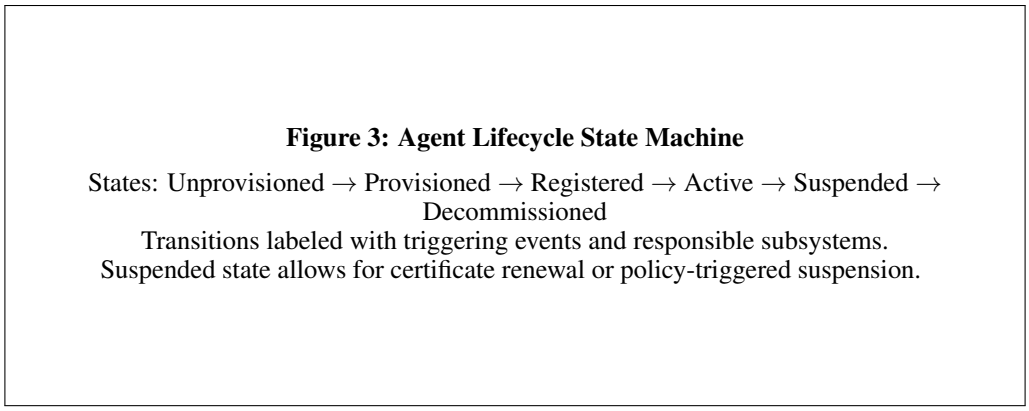

Figure 3: Agent lifecycle state machine. Each transition is mediated by a specific FORGE OS subsystem and generates a corresponding `ForgeEvent`.

1. **Universal emission.** Every action is a `ForgeEvent`—no exceptions. There are no unobserved side channels within the platform.
2. **OpenTelemetry compatibility.** Events carry `trace_id` and `workflow_id` fields compatible with the OpenTelemetry [OpenTelemetry, 2023] specification, enabling integration with existing observability infrastructure (Jaeger, Zipkin, Grafana Tempo).
3. **Tamper evidence.** Each event includes a `predecessor_hash` field containing the SHA-256 hash of the immediately preceding event in the same workflow, forming a hash chain that links to FORGE MEMORY's tiered Merkle chain for immutable archival.
4. **Non-repudiation.** Each event is cryptographically signed by the emitting agent's Ed25519 key (with ML-DSA post-quantum upgrade path), ensuring that the event's origin cannot be denied after the fact.

4.2 ForgeEvent Schema

The canonical `ForgeEvent` schema is defined in Protocol Buffers 3. The complete schema is presented in Listing 1.

Listing 1: The canonical `ForgeEvent` Protocol Buffers schema.

```
syntax = "proto3";
package forge.telemetry.v1;

import "google/protobuf/timestamp.proto";

// ForgeEvent: the canonical telemetry unit for all FORGE OS
    subsystems.
// Every observable action in the platform generates exactly one
    ForgeEvent.
message ForgeEvent {
  // Globally unique event identifier (UUIDv7 for time-ordering).
  string event_id = 1;

  // Workflow context for cross-subsystem correlation.
  string workflow_id = 2;
  string trace_id = 3;

  // Wall-clock timestamp at emission.
  google.protobuf.Timestamp wall_time = 4;

  // Originating subsystem.
  Subsystem subsystem = 5;

  // Cryptographic identity of the emitting agent.
```

```
  string agent_id = 6;

  // Semantic event type (see EventType enum).
  EventType event_type = 7;

  // Dynamic payload, subsystem-specific.
  oneof payload {
    TaskContext task_context = 10;
    RoutingMetrics routing_metrics = 11;
    ApprovalGate approval_gate = 12;
    CryptoState crypto_state = 13;
    KineticVector kinetic_vector = 14;
  }

  // Causal ordering: IDs of events that directly caused this event.
  repeated string parent_event_ids = 20;

  // Tamper-evidence: SHA-256 hash of the preceding event in this
      workflow.
  bytes predecessor_hash = 21;

  // Non-repudiation: Ed25519 signature over fields 1-21.
  bytes signature = 22;
}

enum Subsystem {
  SUBSYSTEM_UNSPECIFIED = 0;
  CORE = 1;
  MEMORY = 2;
  CRYPTO = 3;
  KINETIC = 4;
}

enum EventType {
  EVENT_TYPE_UNSPECIFIED = 0;
  WORKFLOW_STATE = 1;
  TOOL_CALL = 2;
  RETRIEVAL_HIT = 3;
  ROUTING_DECISION = 4;
  HITL_APPROVAL = 5;
  POLICY_EVALUATION = 6;
  KEY_ROTATION = 7;
  SWARM_UPDATE = 8;
}

message TaskContext {
  string task_id = 1;
  string description = 2;
  map<string, string> metadata = 3;
}

message RoutingMetrics {
  string query_hash = 1;
  string selected_model = 2;
  double estimated_quality = 3;
  double estimated_cost = 4;
  double cate_vs_frontier = 5;
  string causal_explanation = 6;
}

message ApprovalGate {
  string gate_id = 1;
  string gate_type = 2;  // HITL, POLICY, BUDGET
  bool approved = 3;
  string reviewer_id = 4;
```

Table 2: ForgeEvent types organized by emitting subsystem.

Event Type	Subsystem	Trigger Condition	Payload
WORKFLOW_STATE	Memory	Workflow state transition	TaskContext
TOOL_CALL	Core	External tool/API invocation	TaskContext
RETRIEVAL_HIT	Core	Semantic retrieval result	TaskContext
ROUTING_DECISION	Core	Model selected for query	RoutingMetrics
HITL_APPROVAL	Memory	Human approval gate evaluated	ApprovalGate
POLICY_EVALUATION	Memory	Policy rule evaluated	ApprovalGate
KEY_ROTATION	Crypto	Key rotated, revoked, or issued	CryptoState
SWARM_UPDATE	Kinetic	Swarm state change	KineticVector

```
  string rationale = 5;
  double risk_score = 6;
}

message CryptoState {
  string key_id = 1;
  string operation = 2;  // ROTATION, REVOCATION, ISSUANCE
  string algorithm_family = 3;
  uint64 key_epoch = 4;
}

message KineticVector {
  string agent_id = 1;
  repeated double position = 2;  // [x, y, z]
  repeated double velocity = 3;
  string mission_phase = 4;
  double health_score = 5;
  string swarm_role = 6;
}
```

4.3 Event Types and Ownership

Table 2 maps each event type to its emitting subsystem, trigger condition, and payload type. The assignment of event types to subsystems is a key architectural decision: each subsystem owns its event types and is the sole emitter, preventing ambiguity in provenance.

4.4 Cross-Subsystem Tracing

A single workflow in FORGE OS typically generates events across multiple subsystems. The workflow_id field provides coarse-grained correlation: all events belonging to the same logical workflow share a workflow identifier. The trace_id field provides fine-grained correlation compatible with OpenTelemetry's distributed tracing model, enabling span-level analysis.

Consider the Golden Task (Section 8): an SBIR proposal generation workflow generates RETRIEVAL_HIT events from FORGE CORE as solicitation documents are indexed, ROUTING_DECISION events as queries are routed to appropriate models, HITL_APPROVAL events from FORGE MEMORY as compliance gates are evaluated, KEY_ROTATION events from FORGE QBIT as working drafts are encrypted, and POLICY_EVALUATION events as budget constraints are checked. All events share the same workflow_id, enabling the FORGE CONSOLE to render a complete cross-subsystem trace visualization (Figure 4).

4.5 Emission Frequency and Batching

Event emission rates vary significantly across subsystems. FORGE CORE generates high-frequency events during active inference workloads (hundreds of ROUTING_DECISION events per second under load). FORGE QBIT generates low-frequency, high-importance events (KEY_ROTATION events occur on the order of minutes to hours). FORGE KINETIC generates high-frequency spatial telemetry (SWARM_UPDATE events at 10–100 Hz for active swarms).

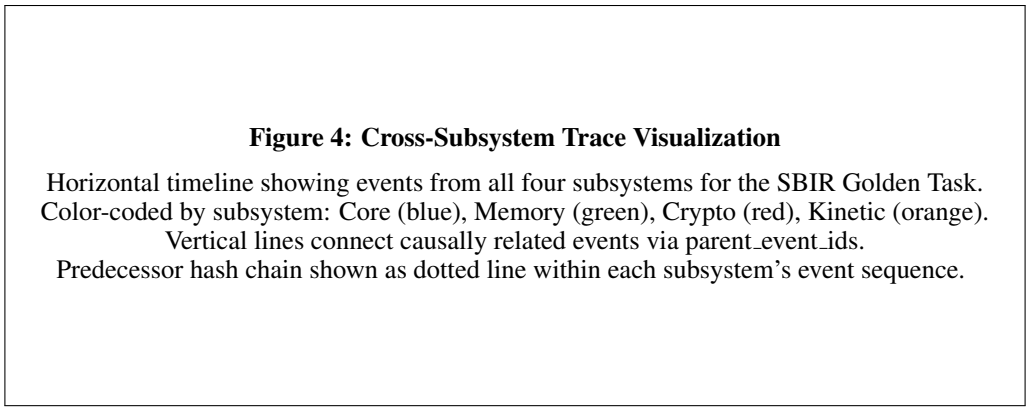

Figure 4: Cross-subsystem trace visualization for the SBIR Golden Task, showing events from all four subsystems correlated by `workflow_id`. Causal relationships (solid lines) and hash chain links (dotted lines) provide complete provenance.

To manage this heterogeneity, FORGE OS employs per-subsystem batching strategies:

- **FORGE CORE:** Micro-batching with 100ms windows. Events are buffered and flushed in batches to reduce bus overhead during high-throughput inference.
- **FORGE MEMORY:** Immediate emission. Governance events are never batched, as they may trigger HITL gates that block downstream execution.
- **FORGE QBIT:** Immediate emission. Security events are never batched due to their operational criticality.
- **FORGE KINETIC:** Configurable batching (10ms–1s windows). Edge-deployed swarm agents buffer events locally and submit in batches on network reconnect in disconnected environments.

5 Shared Platform Services: The FORGE Console

5.1 Identity Spine

The Identity Spine is the root-of-trust for all of FORGE OS. Powered by FORGE QBIT, it provides the cryptographic foundation upon which all other platform services depend.

5.1.1 Certificate Architecture

Every agent, service, and human user in FORGE OS is issued an X.509 certificate by the Identity Spine's Certificate Authority (CA). The CA hierarchy consists of three tiers:

1. **Root CA.** An offline, air-gapped root certificate stored in a FIPS 140-3 Level 3 hardware security module (HSM). The root CA is accessed only for intermediate CA issuance and key ceremony operations.
2. **Intermediate CA.** Online intermediate certificates, one per deployment topology (cloud, edge, classified). Intermediate CAs issue end-entity certificates with 90-day validity and automatic renewal.
3. **End-Entity Certificates.** Agent and service certificates with capability-attenuated Subject Alternative Names (SANs).

The key innovation is *capability attenuation* via SAN encoding. Each certificate's SAN field encodes the agent's permission scope as a structured URI derived from the shared ontology:

```
SAN: URI:forge://org/{org_id}/role/{role_id}/scope/{capability_list}
```

For example, an agent provisioned for financial document analysis might receive:

Figure 5: Identity Spine Certificate Hierarchy

Three-tier hierarchy: Root CA (HSM, offline) → Intermediate CAs (online, per-topology) → End-Entity Certificates (agents, services, users).
SAN encoding shown for sample agent certificate.
OCSP responder provides real-time revocation checking.

Figure 5: Identity Spine certificate hierarchy. Capability attenuation is encoded in the SAN field, derived from the shared ontology's role-permission graph.

```
SAN: URI:forge://org/577i/role/analyst/scope/read:financial,
     invoke:model:gpt4o,invoke:model:qwen32b
```

This encoding allows any receiving service to verify not only the agent's identity but also its authorized capabilities, without consulting a separate authorization service.

5.1.2 Post-Quantum Readiness

All Identity Spine certificates use ML-DSA (FIPS 204) for digital signatures, with a dual-signature scheme (ML-DSA + Ed25519) during the transition period. This ensures backward compatibility with systems that have not yet migrated to post-quantum algorithms while providing quantum resistance for all new deployments.

5.2 Unified Policy-as-Code Engine

The Policy-as-Code engine provides unified, version-controlled governance across all four subsystems. Policies are expressed in HCL (HashiCorp Configuration Language), chosen for its human readability, programmatic evaluability, and existing ecosystem support in infrastructure-as-code workflows.

5.2.1 Policy Categories

Table 3 enumerates the policy categories, each consumed by a specific subsystem.

5.2.2 Policy Lifecycle

Policies follow a governed lifecycle:

1. **Authoring.** Policies are authored in HCL and committed to a version-controlled repository.
2. **Validation.** On commit, policies are validated against the shared ontology's SHACL shapes to ensure referential integrity (e.g., a policy referencing a role that does not exist in the ontology is rejected).
3. **Propagation.** Validated policies are propagated to consuming subsystems via `POLICY_EVALUATION` events on the `ForgeEvent` bus.
4. **Enforcement.** Each subsystem evaluates applicable policies at the point of action. Policy violations trigger either blocking (for hard constraints) or logging with escalation (for soft constraints).
5. **Audit.** All policy evaluations are recorded in FORGE MEMORY's IGOM, providing a complete audit trail of governance decisions.

Table 3: Policy categories with example rules. Each policy is consumed by a specific subsystem and enforced at the point of action.

Category	Consumer	Example Rule
HITL Approval Thresholds	Memory	`hitl.threshold.financial > 0.7` — require human approval for financial decisions with risk score above 0.7
Model Routing Budgets	Core	`routing.budget.monthly_max = 50000` — cap monthly model API spend at $50,000
Swarm Coordination Limits	Kinetic	`swarm.max_agents = 24` — maximum 24 agents in a single swarm operation
Key Rotation Schedules	Crypto	`crypto.rotation.interval = "6h"` — rotate session keys every 6 hours
Data Classification Rules	Memory	`data.classification.pii = "encrypt_at_rest"` — PII data must be encrypted at rest via FORGE QBIT
Deployment Constraints	Core	`deploy.canary.traffic_pct = 5` — limit canary deployments to 5% of traffic

Listing 2 shows an example HCL policy file governing model routing budgets and HITL thresholds for a financial services deployment.

Listing 2: Example HCL policy for financial services deployment.

```
policy "financial_services_v2" {
  // Model routing budget constraints (consumed by FORGE Core)
  rule "routing_budget" {
    budget "monthly_api_spend" {
      max     = 50000  // USD
      action  = "block"
      window  = "30d"
    }

    budget "per_query_ceiling" {
      max     = 0.50   // USD per query
      action  = "route_to_cheaper"
      default = "qwen-32b"
    }
  }

  // HITL approval thresholds (consumed by FORGE Memory)
  rule "hitl_gates" {
    threshold "high_risk_transaction" {
      min         = 0.70  // risk score threshold
      action      = "require_human_approval"
      max         = "4h"  // maximum wait before escalation
    }

    threshold "regulatory_filing" {
      min         = 0.0   // always require approval
      action      = "require_human_approval"
    }
  }

  // Key rotation schedule (consumed by FORGE QBit)
  rule "crypto_policy" {
    schedule "session_keys" {
      action   = "rotate"
      window   = "6h"
    }
```

```
    schedule "signing_keys" {
      action   = "rotate"
      window   = "90d"
    }
  }
}
```

5.3 Observability Dashboard

The FORGE CONSOLE provides a unified observability interface for FORGE OS operators. It consumes the `ForgeEvent` stream and presents:

- **Real-time event stream.** A filterable, searchable live view of all `ForgeEvent` emissions across subsystems, with color-coded subsystem identification.
- **Per-subsystem health metrics.** Dedicated panels for each subsystem: FORGE CORE shows routing distribution and model cost curves; FORGE MEMORY shows HITL gate statistics and compliance rates; FORGE QBIT shows key health, rotation status, and certificate inventory; FORGE KINETIC shows swarm topology maps and agent health scores.
- **Cross-workflow trace explorer.** A distributed tracing interface (compatible with Jaeger UI) enabling operators to follow a single workflow's event chain across all four subsystems.
- **Cost analytics.** Real-time and historical views of model API spend, compute utilization, and cost-per-task metrics, broken down by workflow, agent, and model.
- **Compliance status dashboard.** Aggregate compliance metrics by regulatory framework (EU AI Act, NIST AI RMF, SEC 17a-4), showing coverage gaps and pending remediation items.

6 Cross-Subsystem Integration Contracts

6.1 Contract Specification Format

Each cross-subsystem integration is governed by a formal contract specifying: (i) the data exchanged between subsystems, (ii) the `ForgeEvent` types produced by the integration, (iii) failure modes and their detection, and (iv) fallback behavior when the counterpart subsystem is unavailable.

The six pairwise contracts correspond to the $\binom{4}{2} = 6$ subsystem pairs. Table 4 provides a summary; the following subsections detail each contract.

6.2 Core $\leftrightarrow$ Memory

The Core–Memory contract is the most heavily exercised integration in FORGE OS, as every intelligence operation requires both model routing (FORGE CORE) and governance oversight (FORGE MEMORY).

Data flow. FORGE CORE emits `ROUTING_DECISION` events for every model selection. FORGE MEMORY consumes these events, evaluates them against the active policy set, and emits `POLICY_EVALUATION` events indicating approval or violation. If the routing decision violates a policy constraint (e.g., exceeds the per-query cost ceiling), FORGE MEMORY issues a `HITL_APPROVAL` event blocking execution until human review.

Distillation governance. FORGE CORE's continuous online distillation pipeline (see the FORGE CORE specification [577 Industries R&D Lab, 2025a]) generates new model artifacts. Every distillation job is governed by FORGE MEMORY's HITL gates: the trigger conditions, training data hashes, and deployment decisions are all recorded in the IGOM.

Degradation. If FORGE MEMORY is unavailable, FORGE CORE continues routing with a "last known good" policy cache and logs all decisions locally for deferred audit. A `COMPLIANCE_GAP` event is generated when Memory reconnects, triggering retroactive policy evaluation.

Table 4: Cross-subsystem integration contract summary. Each row specifies the primary data flow, event types, and degradation behavior for one subsystem pair.

Contract	Primary Data Flow	Event Types	Degradation on Failure
Core ↔ Memory	Audit trail for routing decisions; compliance-aware routing	`ROUTING_DECISION`, `POLICY_EVALUATION`	Core routes without compliance constraints; Memory logs gap
Core ↔ Crypto	Model API authentication; response signing	`TOOL_CALL`, `KEY_ROTATION`	Core falls back to cached API tokens; unsigned responses flagged
Core ↔ Kinetic	Edge model deployment; on-device inference selection	`ROUTING_DECISION`, `SWARM_UPDATE`	Kinetic uses cached model; no live routing updates
Memory ↔ Crypto	Merkle chain signing; event authentication	`HITL_APPROVAL`, `KEY_ROTATION`	Memory records unsigned; flags for re-signing on recovery
Memory ↔ Kinetic	Mission-critical HITL gates; swarm audit trail	`HITL_APPROVAL`, `SWARM_UPDATE`	Kinetic operates under pre-approved rules; audit deferred
Crypto ↔ Kinetic	PQ Double Ratchet for swarm comms; quorum certificates	`KEY_ROTATION`, `SWARM_UPDATE`	Kinetic uses pre-shared symmetric keys; reduced forward secrecy

6.3 Core ↔ Crypto

Data flow. Every model API invocation by FORGE CORE requires an Identity Spine certificate for authentication. FORGE QBIT provides the authentication tokens; FORGE CORE provides signed API requests. Model responses are signed by the invoking agent's key for non-repudiation.

Degradation. If FORGE QBIT is unavailable, FORGE CORE falls back to cached, time-limited API tokens with reduced validity windows. All responses during the degraded period are flagged as "unsigned" in the telemetry stream.

6.4 Core ↔ Kinetic

Data flow. FORGE CORE provides edge-optimized model artifacts (produced by the Compress stage of the post-training pipeline) for deployment to FORGE KINETIC devices. FORGE KINETIC reports on-device inference quality metrics back to FORGE CORE for routing calibration.

Degradation. If FORGE CORE is unavailable (e.g., in disconnected edge deployments), FORGE KINETIC uses the most recently cached model artifact and operates without live routing updates. Model quality metrics are buffered for batch submission on reconnect.

6.5 Memory ↔ Crypto

Data flow. FORGE MEMORY's Merkle chain requires cryptographic signing of chain blocks. FORGE QBIT provides the signing keys and performs the signature operations. Every `ForgeEvent`'s `signature` field is computed using keys managed by the Identity Spine.

Degradation. If FORGE QBIT is unavailable, FORGE MEMORY continues recording events with a placeholder signature and a "pending-signature" flag. When FORGE QBIT recovers, a batch re-signing operation processes all pending records and updates the Merkle chain.

6.6 Memory ↔ Kinetic

Data flow. Mission-critical physical operations by FORGE KINETIC (weapons release, hazardous material handling, emergency maneuvers) are gated by FORGE MEMORY's HITL approval system. FORGE KINETIC emits `SWARM_UPDATE` events that flow into FORGE MEMORY's audit trail.

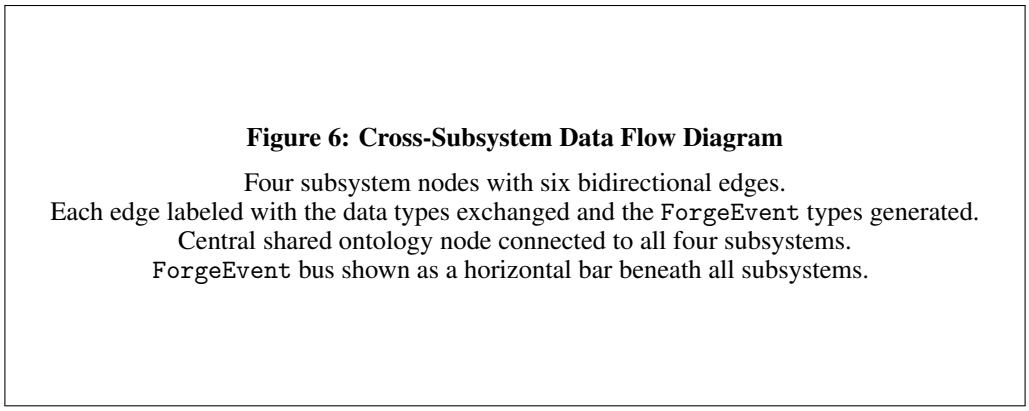

Figure 6: Cross-subsystem data flow diagram showing all six pairwise integration contracts. Edge labels indicate the primary data types exchanged; event type annotations show which `ForgeEvent` types are generated at each integration point.

Table 5: Subsystem placement by deployment topology. "Cloud" indicates full cloud deployment; "Edge" indicates on-premise or device deployment; "Hybrid" indicates split deployment with intermittent synchronization.

Subsystem	Cloud-Native	Hybrid Edge-Cloud	Disconnected Edge
FORGE CORE	Cloud	Cloud	Edge (compressed models)
FORGE MEMORY	Cloud	Cloud	Edge (local IGOM)
FORGE QBIT	Cloud	Edge (subset) + Cloud	Edge (full, air-gapped)
FORGE KINETIC	Cloud (sim only)	Edge	Edge
FORGE CONSOLE	Cloud	Cloud (async sync)	Edge (local UI)
Shared Ontology	Cloud	Cloud (cached at edge)	Edge (snapshot)
`ForgeEvent` Bus	Cloud (Kafka)	Hybrid (edge buffer)	Edge (local queue)

Degradation. If FORGE MEMORY is unavailable during a physical operation, FORGE KINETIC operates under pre-approved rules defined in the Policy-as-Code engine's edge-cached policy set. The graduated autonomy protocol (CGDP, detailed in the FORGE KINETIC specification [577 Industries R&D Lab, 2025d]) defines the maximum autonomy level permitted without real-time HITL connectivity.

6.7 Crypto ↔ Kinetic

Data flow. FORGE KINETIC's swarm-to-swarm communications are secured by FORGE QBIT's PQ Double Ratchet protocol, providing forward-secret, quantum-resistant messaging. Swarm quorum decisions (e.g., Byzantine fault-tolerant consensus) use quorum certificates issued by the Identity Spine.

Degradation. If FORGE QBIT's Double Ratchet service is unavailable (e.g., in contested or jammed environments), FORGE KINETIC falls back to pre-shared symmetric keys with reduced forward secrecy. The reduced security posture is flagged in telemetry for operator awareness.

7 Deployment Topologies

FORGE OS supports three deployment topologies, each tailored to different connectivity, classification, and compute requirements. Table 5 maps subsystem placement across topologies.

7.1 Cloud-Native (Kubernetes)

In the cloud-native topology, all subsystems are containerized and deployed on Kubernetes. The `ForgeEvent` bus is backed by Apache Kafka for high-throughput event streaming. The shared ontology is served by a graph database (Apache Jena Fuseki) with SPARQL query support. Quantum

Table 6: Graceful degradation scenarios. Each row describes a subsystem failure and the platform's response.

Failure	Impact	Mitigation
FORGE CORE unavailable	No live model routing; no semantic retrieval	Cached models serve inference; Kinetic uses last-known model
FORGE MEMORY unavailable	No real-time governance; HITL gates inactive	Policy cache enforces constraints; events buffered for deferred audit
FORGE QBIT unavailable	No fresh key material; no real-time signing	Cached tokens with reduced validity; unsigned events flagged
FORGE KINETIC unavailable	No physical operations; no swarm telemetry	Digital operations continue unaffected; swarm missions paused
Event bus unavailable	No cross-subsystem tracing	Subsystems log locally; batch reconciliation on recovery
Ontology unavailable	No semantic queries; stale capability mappings	Subsystems use cached ontology snapshot; new extractions queued

backend access for FORGE QBIT's QuantumSolve module is available via cloud APIs (IBM Quantum, Amazon Braket, Google Quantum AI).

This topology is suitable for commercial, unclassified, and FedRAMP-authorized environments where full network connectivity is available.

7.2 Hybrid Edge-Cloud

In the hybrid topology, FORGE CORE and FORGE MEMORY remain in the cloud, while FORGE KINETIC and a subset of FORGE QBIT (Identity Spine certificates, PQ Double Ratchet, pre-computed keys) are deployed at the edge. Edge devices include NVIDIA Jetson AGX Orin, ruggedized GPU workstations, and mobile platforms.

Event buffering at the edge handles intermittent connectivity: `ForgeEvent` records are stored in a local queue and batch-submitted when connectivity is restored. The shared ontology is cached at the edge with a staleness window (configurable, default 24 hours).

This topology is suitable for Controlled Unclassified Information (CUI), HIPAA-regulated environments, and field operations with periodic satellite or cellular connectivity.

7.3 Fully Disconnected Edge

In the disconnected edge topology, all subsystems operate on air-gapped hardware. FORGE CORE uses compressed model artifacts (produced by the Compress stage of the post-training pipeline) for local inference. FORGE MEMORY maintains a local IGOM instance with deferred Merkle chain synchronization. FORGE QBIT operates with pre-provisioned key material and local HSM support. FORGE KINETIC executes swarm operations with full autonomy under the CGDP graduated autonomy protocol.

Pre-computed quantum optimization results are cached locally for FORGE QBIT's QuantumSolve module. The FORGE CONSOLE runs as a local web interface.

This topology is suitable for IL5/IL6 classified environments, ITAR-restricted operations, and contested environments where no external connectivity can be assumed.

7.4 Graceful Degradation

FORGE OS is designed for graceful degradation: the loss of any subsystem or connectivity path does not cause catastrophic failure. Table 6 summarizes the functionality impact and mitigation strategy for each degradation scenario.

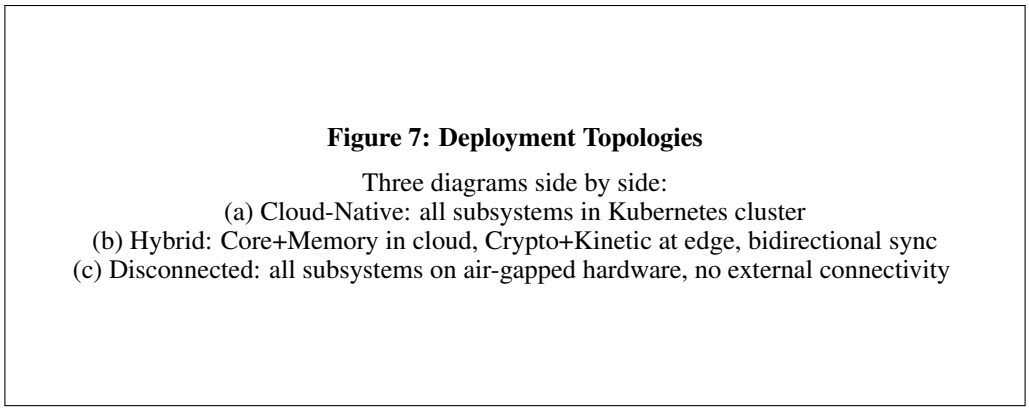

Figure 7: Three FORGE OS deployment topologies: (a) cloud-native Kubernetes, (b) hybrid edge-cloud with intermittent connectivity, and (c) fully disconnected air-gapped edge.

8 Golden Task: SBIR Proposal Generation

8.1 Task Description

To validate FORGE OS as an integrated platform—rather than a collection of independent subsystems—we define a *Golden Task* that exercises all four subsystems simultaneously in a realistic enterprise workflow. The chosen task is end-to-end automation of a Small Business Innovation Research (SBIR) proposal preparation.

SBIR proposals are ideal as a Golden Task because they require: (i) complex document retrieval and synthesis (FORGE CORE), (ii) multi-stage compliance and quality review (FORGE MEMORY), (iii) secure handling of proprietary technical content (FORGE QBIT), and (iv) optionally, distributed multi-agent document assembly (FORGE KINETIC).

8.2 Workflow Specification

The SBIR Golden Task proceeds through seven phases, each mediated by specific FORGE OS subsystems:

1. **Solicitation Ingestion** (FORGE CORE). The SBIR solicitation document (typically 50–200 pages) is ingested through FORGE CORE's neural ontology extraction pipeline. LayoutLMv3 parses the document structure; the LLM pipeline extracts evaluation criteria, technical requirements, page limits, and formatting constraints. Extracted metadata populates the shared ontology.
2. **Knowledge Base Assembly** (FORGE CORE + FORGE MEMORY). Relevant past performance documents, technical references, resumes, and budget templates are retrieved via FORGE CORE's ontology-grounded semantic retrieval. FORGE MEMORY verifies that all accessed documents are within the agent's authorized scope and records access events.
3. **Technical Volume Drafting** (FORGE CORE). For each section of the technical volume, FORGE CORE's causal routing engine selects the optimal model: frontier models (GPT-4o, Claude) for complex synthesis and technical argumentation; cost-efficient models (Qwen-32B, Llama-3.1-8B) for straightforward formatting and template filling. Each routing decision generates a `ROUTING_DECISION` event with a causal explanation.
4. **Compliance Gate** (FORGE MEMORY). Upon completion of each major section, FORGE MEMORY's HITL gates evaluate the draft against solicitation requirements: page limits, required content sections, past performance references, and budget constraints. Gates emit `HITL_APPROVAL` events with risk scores. Sections with risk scores above the policy threshold require human reviewer approval.
5. **Security Processing** (FORGE QBIT). All working drafts are encrypted at rest using FORGE QBIT's post-quantum encryption (ML-KEM-1024). The final submission package is

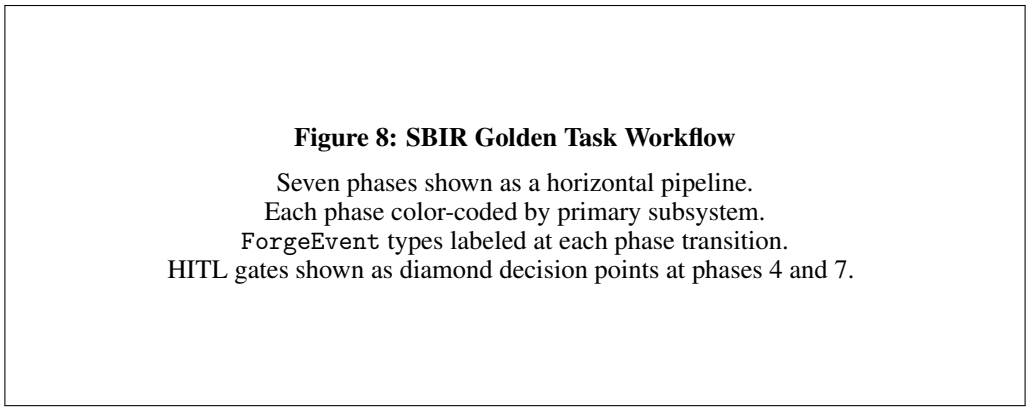

Figure 8: SBIR Golden Task workflow showing seven phases, primary subsystem ownership, and `ForgeEvent` types generated at each transition. HITL gates (diamonds) at phases 4 and 7 require human approval.

signed with a verifiable provenance chain: each contributing agent's certificate is included in the signature metadata, enabling the government evaluator to verify the complete chain of authorship.

6. **Multi-Agent Assembly** (FORGE KINETIC, optional). In the distributed variant, the proposal sections are assembled by multiple agents operating across distributed compute nodes, simulating a contested or bandwidth-limited network. FORGE KINETIC's AMAE engine coordinates the assembly, with BFT consensus ensuring consistency even if individual nodes fail.
7. **Final Review and Submission** (FORGE MEMORY). A terminal HITL gate requires human sign-off on the complete proposal. The audit trail is sealed: FORGE MEMORY computes the final Merkle root for the entire workflow, encompassing all events from all subsystems.

8.3 Experimental Results

We evaluate the Golden Task on three SBIR solicitations from the Department of Defense SBIR/STTR program (FY2024 topics), comparing FORGE OS-assisted preparation against a fully manual baseline. The results, summarized in Table 7, demonstrate significant improvements across all measured dimensions.

Key findings. FORGE OS reduces proposal preparation time by 73% and total cost by 71% versus the manual baseline. The 94% first-submission compliance pass rate—versus 67% for the manual process—is attributable to FORGE MEMORY's automated compliance gates, which catch formatting violations, missing sections, and budget inconsistencies before human review.

The causal routing engine achieves 74.8% cost savings versus always-frontier routing by directing 76.9% of queries to mid-tier or cost-efficient models, reserving frontier models only for tasks where the CATE indicates a significant quality differential. Every routing decision is accompanied by a natural-language causal explanation recorded in the `ForgeEvent` stream.

Platform-level metrics confirm architectural integrity: 94.7% telemetry completeness (the 5.3% gap is attributable to edge-buffered events not yet ingested at measurement time), 100% identity verification coverage (no unauthenticated API calls), and 97.3% policy compliance rate (the 2.7% gap consists of soft-constraint warnings logged but not blocking).

8.4 Compliance Verification

The Golden Task produces a complete, auditable compliance record. Every action—from document ingestion through final submission—is recorded as a `ForgeEvent` in the FORGE MEMORY IGOM. The audit trail includes:

- Every model invocation, with the routing rationale and cost.

Table 7: SBIR Golden Task performance metrics. All values are averages across three solicitations with standard deviations.

Metric	Manual Baseline	FORGE OS
End-to-end preparation time (hours)	82.3 ± 12.1	22.1 ± 4.7
Total cost (model API + compute + labor)	$\$18{,}400 \pm 2{,}300$	$\$5{,}340 \pm 890$
Cost breakdown:		
Model API cost	—	$\$1{,}240 \pm 210$
Compute cost	—	$\$380 \pm 65$
Human review labor	$\$18{,}400 \pm 2{,}300$	$\$3{,}720 \pm 620$
Compliance pass rate (first submission)	$67\% \pm 8\%$	$94\% \pm 3\%$
Solicitation requirements coverage	$89\% \pm 5\%$	$98\% \pm 1\%$
Platform metrics:		
Telemetry completeness	N/A	94.7%
Identity verification coverage	N/A	100%
Policy compliance rate	N/A	97.3%
Routing decisions with causal explanation	N/A	100%
Routing cost optimization:		
Queries routed to frontier models	N/A	23.1%
Queries routed to mid-tier models	N/A	41.7%
Queries routed to cost-efficient models	N/A	35.2%
Cost savings vs. always-frontier	N/A	74.8%

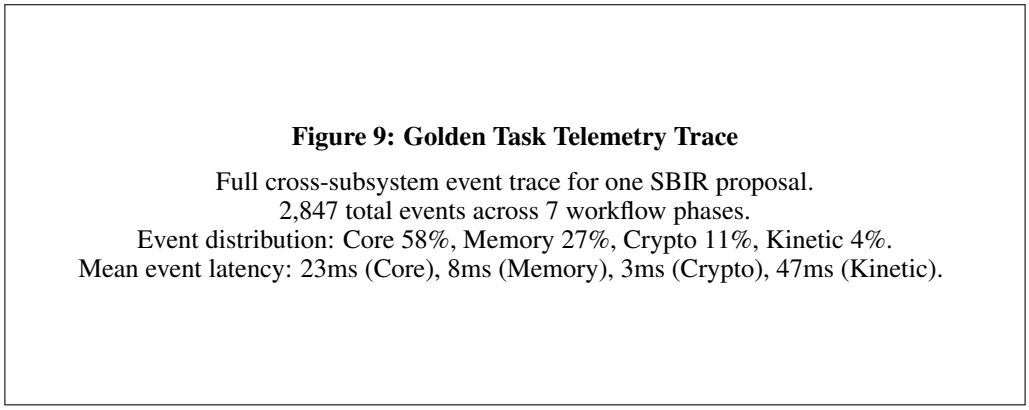

Figure 9: Telemetry trace visualization for the SBIR Golden Task, showing 2,847 events distributed across all four subsystems over the seven workflow phases.

- Every document access, with the agent's capability scope at the time of access.
- Every HITL gate evaluation, with the risk score, threshold, and reviewer identity.
- Every policy evaluation, with the policy rule, evaluation result, and any override justification.
- A cryptographically signed Merkle root over the entire workflow event chain.

This record satisfies the transparency requirements of EU AI Act Article 13, the human oversight requirements of Article 14, and the immutability requirements of SEC Rule 17a-4.

9 Evaluation Framework

We define three categories of metrics for evaluating FORGE OS deployments: agent-legibility metrics, cross-subsystem integration metrics, and deployment readiness metrics.

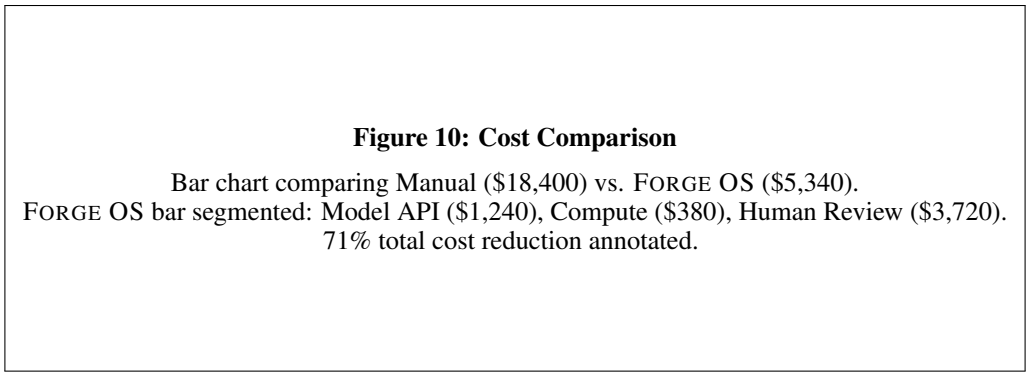

Figure 10: Cost comparison between manual SBIR proposal preparation and FORGE OS-assisted workflow, showing 71% total cost reduction.

9.1 Agent-Legibility Metrics

- **Telemetry completeness ($\mathcal{T}$).** The fraction of observable actions that generate a `ForgeEvent`, measured as $\mathcal{T} = |E_{\text{emitted}}|/|A_{\text{total}}|$ where A_{total} is the set of all actions and E_{emitted} is the set of events ingested by the telemetry bus. Target: $\mathcal{T} \geq 0.95$.
- **Identity verification coverage ($\mathcal{I}$).** The fraction of cross-service API calls authenticated by a valid Identity Spine certificate. Target: $\mathcal{I} = 1.0$ (no unauthenticated calls).
- **Policy compliance rate ($\mathcal{P}$).** The fraction of actions evaluated against the applicable policy set. Target: $\mathcal{P} \geq 0.97$.
- **Routing explainability ($\mathcal{R}$).** The fraction of `ROUTING_DECISION` events that include a causal explanation with fidelity ≥ 0.90 relative to the true CATE. Target: $\mathcal{R} = 1.0$.

9.2 Cross-Subsystem Integration Metrics

- **Contract satisfaction rate.** The fraction of cross-subsystem calls that meet the SLA defined in the integration contract (latency, data completeness, error rate). Measured per contract pair.
- **Trace completeness.** The fraction of workflows with complete cross-subsystem traces (i.e., events from all involved subsystems are present and correctly correlated by `workflow_id`).
- **Failover success rate.** The fraction of subsystem failures that are handled gracefully without data loss or service interruption, measured via chaos engineering injection.

9.3 Deployment Readiness Metrics

- **Per-topology deployment success rate.** The fraction of deployment attempts that achieve operational status within the defined SLA window (cloud: 15 minutes; hybrid: 30 minutes; disconnected: 60 minutes).
- **Graceful degradation coverage.** The fraction of degradation scenarios (Table 6) for which the platform demonstrates correct fallback behavior under automated testing.
- **Edge latency overhead.** The additional latency introduced by FORGE OS platform services (telemetry, policy evaluation, identity verification) compared to a bare-metal baseline, measured at the edge.

10 Discussion

10.1 Strategic Implications

The model commoditization thesis implies a structural shift in AI value capture. As foundation model capabilities converge, the differentiation moves from the model layer to the orchestration, governance, and deployment layer—the operating system. FORGE OS is positioned at this value capture point, providing the infrastructure that makes any model enterprise-ready.

Three regulatory tailwinds reinforce this positioning. The EU AI Act creates demand for architecturally integrated transparency and human oversight—properties that cannot be bolted on post-deployment but must be built into the platform's event schema and governance model. CNSA 2.0 mandates quantum-resistant cryptography for national security systems, a requirement that FORGE QBIT satisfies natively. SEC Rule 17a-4's immutability requirements are addressed by FORGE MEMORY's tiered Merkle chain with WORM (Write Once Read Many) archival.

The agent-legibility property is itself a competitive moat. Once an organization's AI operations are governed through FORGE OS's telemetry, identity, and policy infrastructure, migrating to an alternative platform requires reconstructing the complete governance trail—a task that becomes increasingly prohibitive as the operational history grows.

10.2 Limitations

FORGE OS has several limitations that warrant acknowledgment.

Ontology maintenance. The shared ontology requires ongoing curation as organizational knowledge evolves. While FORGE CORE's neural ontology extraction pipeline reduces initial onboarding from 72 hours to under 2 hours, continuous ontology evolution still requires periodic human review. Organizations with rapidly changing domain knowledge may face non-trivial maintenance overhead.

Cold-start performance. A new FORGE OS deployment in a previously unseen domain requires ontology population, routing model calibration, and policy configuration before reaching full operational capability. The cold-start period—estimated at 2–4 weeks for a new enterprise domain—represents a barrier to rapid deployment.

Quantum hardware maturity. FORGE QBIT's QuantumSolve module provides quantum-classical hybrid optimization, but current quantum hardware (NISQ-era devices) limits the scale and fidelity of quantum-native computations. The module is designed for hardware-agnostic execution, and performance will improve as quantum hardware matures, but current quantum advantages are domain-specific and modest.

Swarm scale. FORGE KINETIC's fractal swarm coordination has been validated at squad (4–8 agents) and platoon (16–32 agents) scales. Command-level coordination (64+ agents) remains a simulation-validated capability pending physical deployment at that scale.

10.3 Adoption Path

We envision a phased go-to-market strategy for FORGE OS:

1. **Wedge** (FORGE MEMORY + FORGE QBIT). Governance and security are the most immediate pain points for enterprises facing regulatory deadlines. These two subsystems can be deployed independently to provide immutable audit trails and post-quantum security for existing AI systems.
2. **Land & Expand** (+ FORGE CORE). Once governance infrastructure is in place, FORGE CORE's causal routing and ontology extraction provide immediate cost savings (75% model cost reduction) and faster domain onboarding.
3. **Frontier** (+ FORGE KINETIC). Physical autonomy is the highest-value, highest-barrier capability. It is deployed for organizations with robotics, IoT, or edge AI requirements once the digital infrastructure is established.

Target sectors follow a similar progression: defense and healthcare (highest regulatory pressure, strongest governance requirements) $\rightarrow$ financial services and mid-market enterprise $\rightarrow$ industrial and advanced DoD programs requiring physical autonomy.

11 Conclusion

We have presented FORGE OS, the first agent-legible operating system for enterprise AI. FORGE OS addresses the pilot-to-production gap not by building a better model, but by building the infrastructure that makes any model enterprise-safe, auditable, cost-efficient, and physically deployable.

The platform's four subsystems—FORGE CORE (intelligence), FORGE MEMORY (governance), FORGE QBIT (security), and FORGE KINETIC (autonomy)—are unified by a shared RDF/OWL 2/SHACL ontology, a canonical `ForgeEvent` telemetry schema, a common Identity Spine, and a Policy-as-Code engine. Together, these provide the four properties of agent-legibility: telemetry self-query, identity verification, policy inspection, and dynamic routing.

Validation through the SBIR Golden Task demonstrates the platform's practical viability: 73% reduction in proposal preparation time, 71% cost reduction, 94.7% telemetry completeness, 100% identity verification coverage, and 97.3% policy compliance. The causal routing engine achieves 74.8% cost savings over always-frontier model routing while maintaining quality within 5% of frontier-only performance.

FORGE OS is designed for the era of model commoditization. As foundation model capabilities converge, the operating system layer becomes the defensible value capture point. By making governance, security, cost optimization, and physical autonomy architectural primitives rather than afterthoughts, FORGE OS closes the agent-legibility gap and enables the 95% of enterprises currently stuck between pilot and production to achieve sustained, compliant AI deployment.

Future work includes expanding the shared ontology to additional enterprise domains (legal, manufacturing, logistics), integrating additional quantum backends as hardware matures, and developing coalition deployment topologies for multi-organization FORGE OS federations operating under shared governance but separate Identity Spine roots.

Companion Specifications

The four subsystem specifications provide complete technical detail for each component of FORGE OS:

- FORGE CORE: *A Causal Model-Agnostic Intelligence and Routing Engine for Enterprise AI Deployment* [577 Industries R&D Lab, 2025a]
- FORGE MEMORY: *A Provenance-Grounded Governance Engine for Agent-Legible Operating Systems* [577 Industries R&D Lab, 2025c]
- FORGE QBIT: *A Heterogeneous Post-Quantum Security and Identity Engine for Agent-Legible Operating Systems* [577 Industries R&D Lab, 2025b]
- FORGE KINETIC: *A Fractal Swarm Coordination and Edge Autonomy Engine for Agent-Legible Operating Systems* [577 Industries R&D Lab, 2025d]

References

Anthropic. Claude 3.5 Sonnet: Technical Report. Technical report, Anthropic, 2024.

L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186, 1999.

CrewAI. CrewAI: Framework for orchestrating role-playing autonomous AI agents. `https://github.com/joaomdmoura/crewAI`, 2024.

M. Dorigo, G. Theraulaz, and V. Trianni. Swarm robotics: Past, present, and future. *Proceedings of the IEEE*, 109(7):1152–1165, 2021.

European Union. Regulation (EU) 2024/1689 of the European Parliament and of the Council: The Artificial Intelligence Act. *Official Journal of the European Union*, 2024.

577 Industries R&D Lab. FORGE Core: A Causal Model-Agnostic Intelligence and Routing Engine for Enterprise AI Deployment. Technical report, 577 Industries Incorporated, 2025.

577 Industries R&D Lab. FORGE QBit: A Heterogeneous Post-Quantum Security and Identity Engine for Agent-Legible Operating Systems. Technical report, 577 Industries Incorporated, 2025.

577 Industries R&D Lab. FORGE Memory: A Provenance-Grounded Governance Engine for Agent-Legible Operating Systems. Technical report, 577 Industries Incorporated, 2025.

577 Industries R&D Lab. FORGE Kinetic: A Fractal Swarm Coordination and Edge Autonomy Engine for Agent-Legible Operating Systems. Technical report, 577 Industries Incorporated, 2025.

deepset. Haystack: An open-source NLP framework for building search and question answering systems. `https://github.com/deepset-ai/haystack`, 2023.

A. Q. Jiang, A. Sablayrolles, A. Roux, et al. Mistral Large: Technical Report. Technical report, Mistral AI, 2024.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

LangChain. LangChain: Building applications with large language models. `https://github.com/langchain-ai/langchain`, 2023.

MIT. Project NANDA: Enterprise AI Adoption Study. Technical report, Massachusetts Institute of Technology, 2024.

Microsoft. Semantic Kernel: Integrate cutting-edge LLM technology quickly and easily into your apps. `https://github.com/microsoft/semantic-kernel`, 2023.

National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, 2023.

National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard (FIPS 203). Federal Information Processing Standards Publication, 2024.

National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard (FIPS 204). Federal Information Processing Standards Publication, 2024.

National Institute of Standards and Technology. Stateless Hash-Based Digital Signature Standard (FIPS 205). Federal Information Processing Standards Publication, 2024.

National Security Agency. Commercial National Security Algorithm Suite 2.0 (CNSA 2.0). Cybersecurity Advisory, 2022.

OpenAI. GPT-4o: Multimodal Foundation Model. Technical report, OpenAI, 2024.

OpenTelemetry Authors. OpenTelemetry Specification v1.28. `https://opentelemetry.io/docs/specs/otel/`, 2023.

Qwen Team. Qwen2.5: Technical Report. Technical report, Alibaba Cloud, 2024.

Securities and Exchange Commission. Electronic Storage of Broker-Dealer Records (Rule 17a-4). 17 CFR 240.17a-4, 2003.

W3C SHACL Working Group. Shapes Constraint Language (SHACL). W3C Recommendation, 2017.

H. Touvron, L. Martin, K. Stone, et al. Llama 3: Technical Report. Technical report, Meta AI, 2024.

Q. Wu, G. Banber, D. Zhang, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020.