<!DOCTYPE html>

```
<h1>Fabric Mind</h1>
<div class="subtitle">Enterprise Cognitive Memory Platform —
Production-grade memory infrastructure for reliable agentic
AI</div>

<h2 id="problem">What Problem Fabric Mind Solves</h2>

<p>Agentic AI systems deployed in enterprise production
environments exhibit systematic failures that stem from a single
architectural gap: the absence of organizational memory.</p>

<p>These systems reason effectively within individual sessions
but cannot learn from outcomes across sessions. Each interaction
begins from zero, with no connection to prior attempts, failures,
or successful resolutions.</p>

<p>The industry response has focused on context engineering—
longer context windows, better retrieval systems, sub-agent
orchestration, and context compaction. These approaches optimize
how agents think at inference time, but they do not address how
systems learn over time.</p>

<p>Context engineering provides agents with more information to
reason over in the moment, but when the session ends, that
context is discarded. The next session starts fresh, repeating
the same diagnostic steps, making the same mistakes, and
rediscovering the same solutions.</p>

<p>This creates four structural failure modes that recur
systematically:</p>

<p><strong>Confident but wrong actions.</strong> Agents execute
decisions with high certainty despite incorrect reasoning because
the system provides no mechanism to compare current reasoning
against past outcomes. Confidence remains decoupled from
accuracy.</p>

<p><strong>Infinite loops and tool thrashing.</strong> Systems
repeat the same failed attempts without recognizing the pattern.
Each retry appears novel to the agent because no attempt history
persists across reasoning cycles.</p>
```

```
<p><strong>Unsafe automation.</strong> Actions execute without awareness of past incidents, constraints, or failure conditions. The agent operates as if every situation is encountered for the first time, ignoring prior escalations or operator interventions.</p>

<p><strong>Reset learning.</strong> Each session starts from zero. Resolved issues recur. Successful patterns are rediscovered repeatedly. The system cannot answer fundamental operational questions: "Have we seen this before?" "What worked last time?" "What usually happens next?"</p>

<div class="callout">
  <div class="callout-title">Core Thesis</div>
  <p style="margin-bottom: 0;">Outcomes, not prompts, drive system learning. Fabric Mind introduces governed organizational memory with explicit gates and outcome feedback loops, transforming agentic AI from stateless reasoning into outcome-calibrated operations.</p>
</div>

<p>Fabric Mind solves this by providing enterprise-grade memory infrastructure that persists experience across sessions, recognizes patterns, enforces safety through gates, and feeds outcomes back to calibrate future behavior. Memory does not replace reasoning—it informs it. Agents still reason over current context, but they do so with awareness of past outcomes, attempt history, and learned patterns.</p>

<p>This is not optional infrastructure. It is the foundation of reliable agentic AI in production.</p>

<div class="callout">
  <div class="callout-title">Positioning</div>
  <p style="margin-bottom: 0;">Fabric Mind is persistent memory infrastructure — not an agent framework or prompt system.</p>
</div>

<h2 id="overview">1. Platform Overview</h2>

<p>Fabric Mind operates as a memory control plane that sits
```

```
between event sources and agentic execution. It captures signals
from enterprise systems, interprets them as impressions, stores
them as durable memory, assembles context for reasoning, detects
patterns, recommends next best actions, gates execution for
safety, and captures outcomes to update memory confidence.</p>

<p>The platform follows a unidirectional flow designed to ensure
every action is informed by past experience and every outcome
reinforces or weakens learned patterns:</p>

<div class="diagram-container">
  <svg width="900" height="520" viewBox="0 0 900 520"
xmlns="http://www.w3.org/2000/svg">
    <!-- Outer frame -->
    <rect x="10" y="10" width="880" height="500" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>

    <!-- Title -->
    <text x="450" y="40" font-size="16" fill="#111827" text-
anchor="middle" font-weight="700">Fabric Mind Platform
Flow</text>

    <!-- Row 1: Ingestion → Impression → Memory Store → Memory
Graph -->
    <rect x="50" y="70" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="130" y="100" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Event & Signal</text>
    <text x="130" y="115" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Ingestion</text>

    <line x1="210" y1="97.5" x2="240" y2="97.5" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="240,97.5 235,94.5 235,100.5"
fill="#6B7280"/>

    <rect x="240" y="70" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="320" y="100" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Impression</text>
    <text x="320" y="115" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Engine</text>
```

```
    <line x1="400" y1="97.5" x2="430" y2="97.5" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="430,97.5 425,94.5 425,100.5"
fill="#6B7280"/>

    <rect x="430" y="70" width="160" height="55" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="510" y="100" font-size="13" fill="#DC2626" text-
anchor="middle" font-weight="700">Memory Store</text>
    <text x="510" y="115" font-size="13" fill="#DC2626" text-
anchor="middle" font-weight="700">(Temporal + Semantic)</text>

    <line x1="590" y1="97.5" x2="620" y2="97.5" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="620,97.5 615,94.5 615,100.5"
fill="#6B7280"/>

    <rect x="620" y="70" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="700" y="100" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Memory</text>
    <text x="700" y="115" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Graph</text>

    <!-- Row 2: Context Assembly → Pattern Detection → Next Best
Action -->
    <line x1="510" y1="125" x2="510" y2="165" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="510,165 507,160 513,160" fill="#6B7280"/>

    <rect x="430" y="165" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="510" y="195" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Context Assembly</text>
    <text x="510" y="210" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Engine</text>

    <line x1="430" y1="192.5" x2="400" y2="192.5"
stroke="#6B7280" stroke-width="2"/>
    <polygon points="400,192.5 405,189.5 405,195.5"
fill="#6B7280"/>
```

```
    <rect x="240" y="165" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="320" y="195" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Pattern Detection</text>
    <text x="320" y="210" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">& Learning</text>

    <line x1="240" y1="192.5" x2="210" y2="192.5"
stroke="#6B7280" stroke-width="2"/>
    <polygon points="210,192.5 215,189.5 215,195.5"
fill="#6B7280"/>

    <rect x="50" y="165" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="130" y="195" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Next Best Action</text>
    <text x="130" y="210" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Engine</text>

    <!-- Row 3: Safety Gate → Action → Outcome -->
    <line x1="130" y1="220" x2="130" y2="260" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="130,260 127,255 133,255" fill="#6B7280"/>

    <rect x="50" y="260" width="160" height="55" fill="#DC2626"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="130" y="285" font-size="13" fill="#FFFFFF" text-
anchor="middle" font-weight="700">GATE: Safety &</text>
    <text x="130" y="300" font-size="13" fill="#FFFFFF" text-
anchor="middle" font-weight="700">Governance</text>

    <line x1="210" y1="287.5" x2="240" y2="287.5"
stroke="#6B7280" stroke-width="2"/>
    <polygon points="240,287.5 235,284.5 235,290.5"
fill="#6B7280"/>

    <rect x="240" y="260" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="320" y="285" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Action Execution</text>
    <text x="320" y="300" font-size="13" fill="#374151" text-
```

```
anchor="middle" font-weight="600">(Gated)</text>

    <line x1="400" y1="287.5" x2="430" y2="287.5"
stroke="#6B7280" stroke-width="2"/>
    <polygon points="430,287.5 425,284.5 425,290.5"
fill="#6B7280"/>

    <rect x="430" y="260" width="160" height="55" fill="#F9FAFB"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="510" y="285" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Outcome</text>
    <text x="510" y="300" font-size="13" fill="#374151" text-
anchor="middle" font-weight="600">Capture</text>

    <!-- Feedback loop: Outcome → Memory Store (orthogonal
routing) -->
    <line x1="590" y1="287.5" x2="810" y2="287.5"
stroke="#DC2626" stroke-width="2" stroke-dasharray="4,3"/>
    <line x1="810" y1="287.5" x2="810" y2="50" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,3"/>
    <line x1="810" y1="50" x2="510" y2="50" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,3"/>
    <line x1="510" y1="50" x2="510" y2="70" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,3"/>
    <polygon points="510,70 507,65 513,65" fill="#DC2626"/>

    <text x="820" y="170" font-size="12" fill="#DC2626" font-
weight="600" font-style="italic">Memory</text>
    <text x="820" y="185" font-size="12" fill="#DC2626" font-
weight="600" font-style="italic">Update</text>

    <!-- Legend -->
    <text x="50" y="380" font-size="12" fill="#6B7280" font-
weight="600">Key Components:</text>
    <rect x="50" y="395" width="14" height="14" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1.5"/>
    <text x="72" y="406" font-size="11" fill="#6B7280">Memory
persistence layer</text>

    <rect x="250" y="395" width="14" height="14" fill="#DC2626"/>
    <text x="272" y="406" font-size="11" fill="#6B7280">Control
gates</text>
```

```
    <line x1="380" y1="402" x2="398" y2="402" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,3"/>
    <text x="406" y="406" font-size="11" fill="#6B7280">Outcome
feedback loop</text>
  </svg>
</div>

<p>Every component in this flow serves a specific operational
purpose. Signals are captured from enterprise systems (incidents,
customer interactions, code changes). Impressions interpret these
signals semantically, extracting meaning and context. Memory
stores these impressions with temporal and semantic indexing,
allowing the system to answer: "Have we seen this before?"
Context assembly retrieves relevant memory and constructs bounded
payloads for reasoning. Pattern detection identifies recurring
situations and learns from outcomes. Next Best Action recommends
responses based on learned patterns. Safety gates enforce
constraints before execution. Outcome capture feeds results back
to memory, reinforcing successful patterns and weakening failed
ones.</p>

<p>This architecture ensures that every action is informed by
past experience and every outcome contributes to future learning.
The system does not rely on prompt engineering or context window
size. It relies on durable memory, explicit gates, and outcome
feedback.</p>

<h2 id="components">2. Core Platform Components</h2>

<p>Each component in the Fabric Mind platform serves a specific
operational function. The following sections describe what each
component does, its inputs and outputs, how it persists data, its
operational behavior, API touchpoints, and how developers
integrate with it.</p>

<h3 id="event-ingestion">2.1 Event & Signal Ingestion</h3>

<p><strong>What it does:</strong> The ingestion layer captures
raw events and signals from enterprise systems—incident alerts,
customer interactions, code commits, deployment events, and
operational metrics. It normalizes these heterogeneous inputs
```

```
into a unified event schema, validates structure, enriches with
metadata (timestamps, source identifiers, tenant context), and
forwards them to the Impression Engine for semantic
interpretation.</p>

<p><strong>Inputs:</strong> JSON payloads from webhooks, message
queues (Kafka, RabbitMQ), HTTP POST endpoints, SDK clients, and
batch uploads. Events include incident alerts, support tickets,
chat transcripts, telemetry streams, and user actions.</p>

<p><strong>Outputs:</strong> Normalized event records with schema
validation, enriched metadata, and correlation identifiers.
Events are routed to the Impression Engine for semantic
processing.</p>

<p><strong>Persistence model:</strong> Events are written to an
append-only log for audit and replay. Retention policies are
configurable per tenant (default 90 days for raw events,
indefinite for derived impressions). Dead-letter queues capture
malformed events for manual review.</p>

<p><strong>Operational behavior:</strong> Ingestion is
horizontally scalable with partitioned queues. Idempotency is
enforced via event IDs to prevent duplicate processing.
Backpressure mechanisms throttle ingest rates if downstream
components are saturated. Failed events are retried with
exponential backoff before moving to dead-letter storage.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/events</code> — Single event ingestion</li>
  <li><code>POST /v1/events/batch</code> — Batch ingestion (up to
1000 events per request)</li>
  <li><code>GET /v1/events/{event_id}</code> — Retrieve event by
ID for audit</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure webhook endpoints in source systems (PagerDuty,
Zendesk, GitHub) to forward events to Fabric Mind ingestion API.
</li>
```

```
  <li>Use SDK clients (Python, Node.js, Go) to emit events programmatically from custom applications.</li>
  <li>Set up batch ingestion jobs for historical data backfill or periodic synchronization.</li>
  <li>Monitor ingestion metrics (event rate, validation errors, dead-letter queue depth) via observability dashboards.</li>
  <li>Define retention policies and schema validation rules per event type in tenant configuration.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="270" viewBox="0 0 760 270" xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="250" fill="#FFFFFF" stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="111827" text-anchor="middle" font-weight="700">Event & Signal Ingestion</text>

    <!-- Event Sources -->
    <rect x="40" y="60" width="120" height="40" fill="#F3F4F6" stroke="#9CA3AF" stroke-width="1.5" rx="4"/>
    <text x="100" y="85" font-size="11" fill="#4B5563" text-anchor="middle" font-weight="600">Incident Alerts</text>

    <rect x="40" y="110" width="120" height="40" fill="#F3F4F6" stroke="#9CA3AF" stroke-width="1.5" rx="4"/>
    <text x="100" y="135" font-size="11" fill="#4B5563" text-anchor="middle" font-weight="600">Support Tickets</text>

    <rect x="40" y="160" width="120" height="40" fill="#F3F4F6" stroke="#9CA3AF" stroke-width="1.5" rx="4"/>
    <text x="100" y="185" font-size="11" fill="#4B5563" text-anchor="middle" font-weight="600">Code Commits</text>

    <rect x="40" y="210" width="120" height="40" fill="#F3F4F6" stroke="#9CA3AF" stroke-width="1.5" rx="4"/>
    <text x="100" y="235" font-size="11" fill="#4B5563" text-anchor="middle" font-weight="600">Telemetry</text>

    <!-- Arrows to Ingestion -->
    <line x1="160" y1="80" x2="240" y2="140" stroke="#6B7280" stroke-width="1.5"/>
```

```
    <line x1="160" y1="130" x2="240" y2="140" stroke="#6B7280"
stroke-width="1.5"/>
    <line x1="160" y1="180" x2="240" y2="140" stroke="#6B7280"
stroke-width="1.5"/>
    <line x1="160" y1="230" x2="240" y2="140" stroke="#6B7280"
stroke-width="1.5"/>

    <!-- Ingestion Layer -->
    <rect x="240" y="100" width="160" height="80" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="320" y="130" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Ingestion Layer</text>
    <text x="320" y="148" font-size="10" fill="#3B82F6" text-
anchor="middle">• Schema validation</text>
    <text x="320" y="162" font-size="10" fill="#3B82F6" text-
anchor="middle">• Metadata enrichment</text>
    <text x="320" y="176" font-size="10" fill="#3B82F6" text-
anchor="middle">• Idempotency check</text>

    <!-- Arrow to Append-Only Log -->
    <line x1="400" y1="140" x2="480" y2="140" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="480,140 475,137 475,143" fill="#6B7280"/>

    <!-- Append-Only Log -->
    <rect x="480" y="100" width="140" height="80" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="550" y="130" font-size="12" fill="#DC2626" text-
anchor="middle" font-weight="700">Append-Only Log</text>
    <text x="550" y="148" font-size="10" fill="#991B1B" text-
anchor="middle">Audit & Replay</text>
    <text x="550" y="162" font-size="10" fill="#991B1B" text-
anchor="middle">Retention: 90 days</text>
    <text x="550" y="176" font-size="10" fill="#991B1B" text-
anchor="middle">Dead-letter queue</text>

    <!-- Arrow to Impression Engine -->
    <line x1="620" y1="140" x2="680" y2="140" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="680,140 675,137 675,143" fill="#6B7280"/>

    <text x="700" y="145" font-size="10" fill="#6B7280" font-
```

```
weight="600">To Impression</text>
    <text x="700" y="158" font-size="10" fill="#6B7280" font-
weight="600">Engine</text>
  </svg>
</div>

<h3 id="impression-engine">2.2 Impression Engine</h3>

<p><strong>What it does:</strong> The Impression Engine
transforms raw events into semantic impressions—structured
interpretations that capture meaning, intent, and context. It
extracts entities, identifies situation types, computes semantic
embeddings, and tags impressions with metadata that enables
downstream pattern detection and context assembly. This is where
unstructured signals become queryable memory.</p>

<p><strong>Inputs:</strong> Normalized events from the ingestion
layer, including incident descriptions, support ticket text, chat
transcripts, and telemetry annotations.</p>

<p><strong>Outputs:</strong> Semantic impressions with extracted
entities (service names, error codes, user IDs), situation
signatures (incident type, severity, triggering conditions),
embeddings for similarity search, and temporal markers.</p>

<p><strong>Persistence model:</strong> Impressions are written to
the Memory Store with both temporal and semantic indexes. Each
impression includes provenance (source event ID, timestamp,
processing version) and is immutable once written. Retention
follows memory decay policies based on recency and reinforcement.
</p>

<p><strong>Operational behavior:</strong> The engine uses
embedding models (sentence transformers, domain-specific fine-
tuned models) to generate semantic vectors. Entity extraction
leverages NER models and domain-specific lexicons. Processing is
idempotent and can be rerun with updated models to regenerate
impressions. Failures are logged and retried; malformed inputs
are flagged for review.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
```

```
  <li><code>POST /v1/impressions/generate</code> — Generate impression from event (internal API)</li>
  <li><code>GET /v1/impressions/{impression_id}</code> — Retrieve impression by ID</li>
  <li><code>POST /v1/impressions/search</code> — Semantic search across impressions</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure entity extraction rules and domain lexicons for specific use cases (e.g., service names, error patterns).</li>
  <li>Select or fine-tune embedding models for semantic similarity (default: all-MiniLM-L6-v2 for general use).</li>
  <li>Define situation taxonomies (incident types, interaction categories) to tag impressions consistently.</li>
  <li>Use semantic search API to query historical impressions by similarity to current events.</li>
  <li>Monitor impression quality metrics (entity extraction accuracy, embedding drift) and retrain models as needed.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="300" viewBox="0 0 760 300" xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="280" fill="#FFFFFF" stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-anchor="middle" font-weight="700">Impression Engine</text>

    <!-- Raw Event -->
    <rect x="40" y="60" width="140" height="60" fill="#F9FAFB" stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="110" y="85" font-size="11" fill="#374151" text-anchor="middle" font-weight="600">Raw Event</text>
    <text x="110" y="102" font-size="9" fill="#6B7280" text-anchor="middle">"Service X degraded"</text>

    <!-- Arrow -->
    <line x1="180" y1="90" x2="220" y2="90" stroke="#6B7280" stroke-width="1.5"/>
    <polygon points="220,90 215,87 215,93" fill="#6B7280"/>
```

```
    <!-- Processing Steps -->
    <rect x="220" y="60" width="180" height="160" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="310" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Semantic Processing</text>

    <rect x="235" y="100" width="150" height="30" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="310" y="120" font-size="10" fill="#1E40AF" text-
anchor="middle">Entity Extraction</text>

    <rect x="235" y="138" width="150" height="30" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="310" y="158" font-size="10" fill="#1E40AF" text-
anchor="middle">Situation Classification</text>

    <rect x="235" y="176" width="150" height="30" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="310" y="196" font-size="10" fill="#1E40AF" text-
anchor="middle">Embedding Generation</text>

    <!-- Arrow -->
    <line x1="400" y1="140" x2="460" y2="140" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="460,140 455,137 455,143" fill="#6B7280"/>

    <!-- Semantic Impression -->
    <rect x="460" y="60" width="260" height="160" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="590" y="85" font-size="12" fill="#DC2626" text-
anchor="middle" font-weight="700">Semantic Impression</text>
    <text x="590" y="108" font-size="9" fill="#991B1B" text-
anchor="middle">Entities: [service_x, degradation, config_change]
</text>
    <text x="590" y="124" font-size="9" fill="#991B1B" text-
anchor="middle">Situation:
incident_performance_degradation</text>
    <text x="590" y="140" font-size="9" fill="#991B1B" text-
anchor="middle">Embedding: [0.23, -0.41, 0.67, ...]</text>
    <text x="590" y="156" font-size="9" fill="#991B1B" text-
anchor="middle">Timestamp: 2026-01-09T10:15:00Z</text>
```

```
    <text x="590" y="172" font-size="9" fill="#991B1B" text-
anchor="middle">Provenance: event_abc123</text>
    <text x="590" y="200" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">→ To Memory Store</text>
  </svg>
</div>

<h3 id="memory-store">2.3 Memory Store (Temporal + Semantic)</h3>

<p><strong>What it does:</strong> The Memory Store persists
semantic impressions with dual indexing: temporal (when did this
happen) and semantic (what does this mean). It enables the system
to answer two critical questions: "Have we seen this before?"
(semantic similarity search) and "When was this relevant?"
(temporal decay and freshness). The store implements memory decay
policies, reinforcement mechanisms, and provenance tracking to
ensure that memory influences reasoning appropriately over time.
</p>

<p><strong>Inputs:</strong> Semantic impressions from the
Impression Engine, including embeddings, entities, situation
signatures, timestamps, and provenance metadata.</p>

<p><strong>Outputs:</strong> Query results for semantic
similarity searches (top-k similar impressions), temporal range
queries (impressions within time windows), and memory manifests
(collections of relevant impressions with confidence scores and
freshness indicators).</p>

<p><strong>Persistence model:</strong> Impressions are stored in
a vector database with semantic embeddings (for similarity
search) and a time-series store with temporal indexes (for
recency queries). Each impression includes decay metadata (last
accessed, reinforcement count, confidence score) that degrades
over time unless reinforced by repeated exposure or successful
outcomes. Retention policies are tenant-configurable with default
2-year retention for reinforced memories and 90-day retention for
unreinforced impressions.</p>

<p><strong>Operational behavior:</strong> Writes are append-only
with versioning. Updates create new versions rather than
modifying existing records. Decay is computed lazily at query
```

```
time based on elapsed time since last access and reinforcement
count. Similarity searches use approximate nearest neighbor (ANN)
algorithms for sub-100ms latency at scale. The store supports
multi-tenant isolation with per-tenant encryption keys and access
controls. Backup and disaster recovery follow enterprise SLOs
with RPO < 1 hour and RTO < 4 hours.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/memory/write</code> — Write impression to
memory store</li>
  <li><code>POST /v1/memory/search/semantic</code> — Semantic
similarity search by embedding</li>
  <li><code>POST /v1/memory/search/temporal</code> — Temporal
range query</li>
  <li><code>POST /v1/memory/reinforce</code> — Reinforce memory
based on outcome</li>
  <li><code>GET /v1/memory/{memory_id}</code> — Retrieve specific
memory by ID</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure retention policies and decay parameters per
tenant (default: 2-year retention, exponential decay with 30-day
half-life).</li>
  <li>Write impressions to memory store immediately after
semantic processing in the Impression Engine.</li>
  <li>Query semantic similarity during context assembly to
retrieve relevant past experiences.</li>
  <li>Query temporal ranges to filter memories by recency or
specific time windows (e.g., "incidents in the last 7 days").
</li>
  <li>Reinforce memories after successful outcomes to increase
confidence and extend retention.</li>
  <li>Monitor memory store metrics (write throughput, query
latency, storage utilization, decay rate) via observability
dashboards.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="340" viewBox="0 0 760 340"
```

```
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="320" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">Memory Store (Temporal +
Semantic)</text>

    <!-- Impression Input -->
    <rect x="40" y="60" width="160" height="60" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1.5" rx="4"/>
    <text x="120" y="85" font-size="11" fill="#DC2626" text-
anchor="middle" font-weight="600">Semantic Impression</text>
    <text x="120" y="102" font-size="9" fill="#991B1B" text-
anchor="middle">Embedding + Entities</text>

    <!-- Arrow -->
    <line x1="200" y1="90" x2="240" y2="90" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="240,90 235,87 235,93" fill="#6B7280"/>

    <!-- Dual Indexing -->
    <rect x="240" y="60" width="240" height="180" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="360" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Dual Indexing</text>

    <!-- Semantic Index -->
    <rect x="260" y="100" width="200" height="60" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="360" y="120" font-size="10" fill="#1E40AF" text-
anchor="middle" font-weight="600">Semantic Index</text>
    <text x="360" y="135" font-size="9" fill="#3B82F6" text-
anchor="middle">Vector DB (ANN search)</text>
    <text x="360" y="148" font-size="9" fill="#3B82F6" text-
anchor="middle">"Have we seen this before?"</text>

    <!-- Temporal Index -->
    <rect x="260" y="170" width="200" height="60" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="360" y="190" font-size="10" fill="#1E40AF" text-
anchor="middle" font-weight="600">Temporal Index</text>
    <text x="360" y="205" font-size="9" fill="#3B82F6" text-
```

```
anchor="middle">Time-series store</text>
    <text x="360" y="218" font-size="9" fill="#3B82F6" text-
anchor="middle">"When was this relevant?"</text>

    <!-- Arrows to Queries -->
    <line x1="480" y1="130" x2="520" y2="100" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="520,100 518,105 515,100" fill="#6B7280"/>

    <line x1="480" y1="200" x2="520" y2="180" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="520,180 518,185 515,180" fill="#6B7280"/>

    <!-- Query Results -->
    <rect x="520" y="60" width="200" height="60" fill="#F3F4F6"
stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="620" y="85" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">Similarity Search
Results</text>
    <text x="620" y="100" font-size="9" fill="#6B7280" text-
anchor="middle">Top-k similar impressions</text>

    <rect x="520" y="140" width="200" height="60" fill="#F3F4F6"
stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="620" y="165" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">Temporal Range Results</text>
    <text x="620" y="180" font-size="9" fill="#6B7280" text-
anchor="middle">Impressions within time window</text>

    <!-- Decay & Reinforcement -->
    <rect x="240" y="260" width="240" height="60" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1.5" rx="4"/>
    <text x="360" y="280" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">Decay & Reinforcement</text>
    <text x="360" y="295" font-size="9" fill="#991B1B" text-
anchor="middle">Confidence degrades over time</text>
    <text x="360" y="308" font-size="9" fill="#991B1B" text-
anchor="middle">Successful outcomes reinforce memory</text>
  </svg>
</div>

<h3 id="memory-graph">2.4 Memory Graph</h3>
```

```
<p><strong>What it does:</strong> The Memory Graph connects related impressions into a navigable knowledge structure, capturing relationships between events, entities, outcomes, and patterns. It enables the system to traverse memory contextually—following causal chains ("this led to that"), entity relationships ("these incidents involved the same service"), and outcome patterns ("these actions produced similar results"). The graph supports root cause analysis, pattern discovery, and context-aware reasoning by making implicit connections explicit. </p>

<p><strong>Inputs:</strong> Impressions from the Memory Store with extracted entities, situation signatures, and outcome metadata. Relationship hints from pattern detection (co-occurrence, temporal proximity, causal links).</p>

<p><strong>Outputs:</strong> Graph traversal results (connected impressions following specified relationship types), subgraph extractions (clusters of related memories), and relationship strength scores (confidence in connections based on reinforcement and recency).</p>

<p><strong>Persistence model:</strong> Graph structure is stored in a graph database with nodes (impressions) and edges (relationships). Edges are typed (causal, co-occurrence, entity-based, outcome-based) and weighted (relationship strength decays over time unless reinforced). Graph snapshots are versioned for rollback and audit. Tenant isolation ensures graph queries cannot traverse across tenant boundaries.</p>

<p><strong>Operational behavior:</strong> Graph construction is incremental—new impressions are added as nodes, and relationships are inferred based on entity overlap, temporal proximity, and outcome similarity. Relationship inference runs asynchronously after impression writes. Graph queries use bounded traversal depth (default: 3 hops) to prevent runaway expansion. Relationship strength is recomputed periodically based on decay policies. The graph supports read-heavy workloads with caching and materialized views for common traversal patterns.</p>

<p><strong>API touchpoints:</strong></p>
```

```
<ul>
  <li><code>POST /v1/graph/traverse</code> — Traverse graph from starting impression</li>
  <li><code>POST /v1/graph/subgraph</code> — Extract subgraph around entity or situation</li>
  <li><code>GET /v1/graph/relationships/{impression_id}</code> — Get all relationships for impression</li>
  <li><code>POST /v1/graph/infer</code> — Trigger relationship inference (internal API)</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure relationship inference rules (entity overlap threshold, temporal proximity window, outcome similarity threshold).</li>
  <li>Use graph traversal during context assembly to retrieve not just similar impressions, but also causally related ones.</li>
  <li>Extract subgraphs for root cause analysis (e.g., "show all incidents related to service X in the last 30 days").</li>
  <li>Query relationships to understand why certain memories are connected (provenance of inferred links).</li>
  <li>Monitor graph metrics (node count, edge count, traversal latency, relationship inference throughput) via observability dashboards.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="320" viewBox="0 0 760 320" xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="300" fill="#FFFFFF" stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-anchor="middle" font-weight="700">Memory Graph</text>

    <!-- Central Node -->
    <circle cx="380" cy="160" r="40" fill="#FEF2F2" stroke="#DC2626" stroke-width="2"/>
    <text x="380" y="155" font-size="10" fill="#DC2626" text-anchor="middle" font-weight="600">Incident A</text>
    <text x="380" y="170" font-size="8" fill="#991B1B" text-anchor="middle">Service X down</text>
```

```
    <!-- Related Nodes -->
    <circle cx="240" cy="100" r="35" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="1.5"/>
    <text x="240" y="100" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">Incident B</text>
    <text x="240" y="112" font-size="7" fill="#3B82F6" text-
anchor="middle">Config change</text>

    <circle cx="520" cy="100" r="35" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="1.5"/>
    <text x="520" y="100" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">Incident C</text>
    <text x="520" y="112" font-size="7" fill="#3B82F6" text-
anchor="middle">Same service</text>

    <circle cx="240" cy="220" r="35" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="1.5"/>
    <text x="240" y="220" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">Resolution</text>
    <text x="240" y="232" font-size="7" fill="#3B82F6" text-
anchor="middle">Rollback</text>

    <circle cx="520" cy="220" r="35" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="1.5"/>
    <text x="520" y="220" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">Pattern</text>
    <text x="520" y="232" font-size="7" fill="#3B82F6" text-
anchor="middle">Recurring</text>

    <!-- Relationships (Edges) -->
    <line x1="275" y1="115" x2="345" y2="145" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,2"/>
    <text x="310" y="125" font-size="8" fill="#DC2626" font-
weight="600">causal</text>

    <line x1="415" y1="145" x2="485" y2="115" stroke="#6B7280"
stroke-width="1.5"/>
    <text x="450" y="125" font-size="8"
fill="#6B7280">entity</text>

    <line x1="345" y1="175" x2="275" y2="205" stroke="#10B981"
```

```
stroke-width="1.5"/>
    <text x="310" y="195" font-size="8" fill="#10B981" font-
weight="600">outcome</text>

    <line x1="415" y1="175" x2="485" y2="205" stroke="#6B7280"
stroke-width="1.5"/>
    <text x="450" y="195" font-size="8"
fill="#6B7280">temporal</text>

    <!-- Legend -->
    <rect x="40" y="260" width="680" height="40" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="50" y="278" font-size="9" fill="#374151" font-
weight="600">Relationship Types:</text>
    <line x1="140" y1="275" x2="160" y2="275" stroke="#DC2626"
stroke-width="2" stroke-dasharray="4,2"/>
    <text x="165" y="278" font-size="8"
fill="#6B7280">Causal</text>
    <line x1="210" y1="275" x2="230" y2="275" stroke="#6B7280"
stroke-width="1.5"/>
    <text x="235" y="278" font-size="8"
fill="#6B7280">Entity</text>
    <line x1="280" y1="275" x2="300" y2="275" stroke="#10B981"
stroke-width="1.5"/>
    <text x="305" y="278" font-size="8"
fill="#6B7280">Outcome</text>
    <line x1="360" y1="275" x2="380" y2="275" stroke="#6B7280"
stroke-width="1.5"/>
    <text x="385" y="278" font-size="8"
fill="#6B7280">Temporal</text>
    <text x="50" y="293" font-size="8" fill="#6B7280">Graph
enables: Root cause analysis, pattern discovery, context-aware
reasoning</text>
  </svg>
</div>

<h3 id="context-assembly">2.5 Context Assembly Engine</h3>

<p><strong>What it does:</strong> The Context Assembly Engine
constructs bounded, relevant context payloads for reasoning
engines by querying memory, filtering for freshness and
confidence, and packaging results into a structured manifest. It
```

```
answers the question: "What does the agent need to know right
now?" by retrieving semantically similar impressions, traversing
the memory graph for causal context, applying freshness gates,
and assembling a compact payload that fits within reasoning token
limits while maximizing relevance and provenance.</p>

<p><strong>Inputs:</strong> Current situation signature (event,
entities, context), query parameters (similarity threshold,
temporal window, max results), and tenant-specific assembly
policies (freshness requirements, confidence thresholds, graph
traversal depth).</p>

<p><strong>Outputs:</strong> Context manifest containing:
relevant impressions (ranked by similarity and freshness), graph-
connected memories (causally related impressions), confidence
scores (per impression), provenance metadata (timestamps,
sources, reinforcement counts), and assembly metadata (query
parameters, result count, truncation indicators).</p>

<p><strong>Persistence model:</strong> Context assembly is
stateless and ephemeral—manifests are not persisted by default.
Assembly requests and results are logged for audit and debugging.
Assembly policies (tenant-specific rules for freshness,
confidence, graph depth) are stored in tenant configuration.</p>

<p><strong>Operational behavior:</strong> Assembly is synchronous
with sub-200ms latency target. The engine queries memory store
(semantic + temporal), traverses memory graph (bounded depth),
applies freshness and confidence filters, ranks results by
relevance, and truncates to fit token budget. Assembly is
idempotent—same input produces same output given unchanged memory
state. Failures are retried with exponential backoff; degraded
mode returns partial results with warnings if memory queries
timeout.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/context/assemble</code> — Assemble context
for current situation</li>
  <li><code>POST /v1/context/preview</code> — Preview context
without logging (for testing)</li>
  <li><code>GET /v1/context/policies</code> — Retrieve tenant
```

```
assembly policies</li>
  <li><code>PUT /v1/context/policies</code> — Update tenant
assembly policies</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure assembly policies per tenant (freshness window:
30 days, confidence threshold: 0.6, max results: 20, graph depth:
2 hops).</li>
  <li>Call context assembly before invoking reasoning engine,
passing current situation signature and query parameters.</li>
  <li>Inject assembled context manifest into reasoning prompt as
structured data (JSON or formatted text).</li>
  <li>Use preview endpoint during development to test assembly
behavior without affecting audit logs.</li>
  <li>Monitor assembly metrics (latency, result count, truncation
rate, freshness distribution) via observability dashboards.</li>
  <li>Tune assembly policies based on reasoning outcomes
(increase freshness window if too few results, decrease if too
many stale memories).</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="360" viewBox="0 0 760 360"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="340" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">Context Assembly Engine</text>

    <!-- Current Situation -->
    <rect x="40" y="60" width="140" height="60" fill="#F3F4F6"
stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="110" y="85" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">Current Situation</text>
    <text x="110" y="100" font-size="8" fill="#6B7280" text-
anchor="middle">Event + Entities</text>

    <!-- Arrow -->
    <line x1="180" y1="90" x2="220" y2="90" stroke="#6B7280"
stroke-width="1.5"/>
```

```
    <polygon points="220,90 215,87 215,93" fill="#6B7280"/>

    <!-- Assembly Process -->
    <rect x="220" y="60" width="320" height="240" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="380" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Assembly Process</text>

    <!-- Step 1 -->
    <rect x="240" y="100" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="120" font-size="9" fill="#1E40AF" text-
anchor="middle">1. Query Memory Store (semantic + temporal)
</text>

    <!-- Step 2 -->
    <rect x="240" y="145" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="165" font-size="9" fill="#1E40AF" text-
anchor="middle">2. Traverse Memory Graph (bounded depth)</text>

    <!-- Step 3 -->
    <rect x="240" y="190" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="210" font-size="9" fill="#1E40AF" text-
anchor="middle">3. Apply Freshness & Confidence Filters</text>

    <!-- Step 4 -->
    <rect x="240" y="235" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="255" font-size="9" fill="#1E40AF" text-
anchor="middle">4. Rank & Truncate to Token Budget</text>

    <!-- Arrow -->
    <line x1="540" y1="180" x2="580" y2="180" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="580,180 575,177 575,183" fill="#6B7280"/>

    <!-- Context Manifest -->
    <rect x="580" y="60" width="140" height="240" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="4"/>
    <text x="650" y="85" font-size="10" fill="#DC2626" text-
```

```
anchor="middle" font-weight="600">Context Manifest</text>
    <text x="650" y="108" font-size="8" fill="#991B1B" text-
anchor="middle">Relevant impressions</text>
    <text x="650" y="123" font-size="8" fill="#991B1B" text-
anchor="middle">Graph-connected</text>
    <text x="650" y="138" font-size="8" fill="#991B1B" text-
anchor="middle">memories</text>
    <text x="650" y="158" font-size="8" fill="#991B1B" text-
anchor="middle">Confidence scores</text>
    <text x="650" y="173" font-size="8" fill="#991B1B" text-
anchor="middle">Provenance metadata</text>
    <text x="650" y="193" font-size="8" fill="#991B1B" text-
anchor="middle">Assembly metadata</text>
    <text x="650" y="218" font-size="9" fill="#DC2626" text-
anchor="middle" font-weight="600">Bounded & Relevant</text>
    <text x="650" y="233" font-size="8" fill="#991B1B" text-
anchor="middle">Fits token budget</text>
    <text x="650" y="248" font-size="8" fill="#991B1B" text-
anchor="middle">Maximizes relevance</text>

    <!-- Target Latency -->
    <rect x="220" y="310" width="320" height="30" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="380" y="330" font-size="9" fill="#374151" text-
anchor="middle" font-weight="600">Target Latency: &lt; 200ms |
Idempotent | Degraded mode on timeout</text>
  </svg>
</div>

<h3 id="pattern-detection">2.6 Pattern Detection & Learning</h3>

<p><strong>What it does:</strong> The Pattern Detection &
Learning component identifies recurring situations, outcome
correlations, and behavioral trends across accumulated memory. It
enables the system to learn from experience by detecting patterns
such as "configuration changes on Fridays lead to incidents,"
"rollback actions resolve 80% of service outages," or "customers
asking about feature X are likely to churn." Patterns are
represented as probabilistic rules with confidence scores,
reinforcement counts, and decay mechanisms to ensure learning
adapts to changing conditions.</p>
```

```
<p><strong>Inputs:</strong> Impressions from Memory Store with outcome labels (success, failure, neutral), entity co-occurrence data from Memory Graph, and temporal sequences (event chains over time).</p>

<p><strong>Outputs:</strong> Detected patterns (situation → outcome correlations with confidence scores), pattern manifests (collections of related patterns for specific situations), and pattern evolution metrics (how pattern confidence changes over time based on reinforcement or contradiction).</p>

<p><strong>Persistence model:</strong> Patterns are stored as probabilistic rules with metadata: situation signature (entities, context), outcome distribution (success rate, failure rate), confidence score (based on sample size and consistency), reinforcement count (how many times pattern has been observed), and last updated timestamp. Patterns decay over time unless reinforced by new observations. Pattern versioning enables rollback if patterns become stale or misleading.</p>

<p><strong>Operational behavior:</strong> Pattern detection runs asynchronously as a background process, analyzing batches of impressions for co-occurrence, temporal sequences, and outcome correlations. Detection uses statistical thresholds (minimum sample size: 10, minimum confidence: 0.7) to avoid spurious patterns. Patterns are updated incrementally as new impressions arrive—reinforced if consistent, weakened if contradicted. Pattern queries are read-heavy with caching for frequently accessed patterns. The system supports pattern freezing (lock patterns for audit or compliance) and pattern rollback (revert to previous pattern state).</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/patterns/detect</code> — Trigger pattern detection (internal API)</li>
  <li><code>POST /v1/patterns/query</code> — Query patterns for situation</li>
  <li><code>GET /v1/patterns/{pattern_id}</code> — Retrieve specific pattern</li>
  <li><code>POST /v1/patterns/freeze</code> — Freeze pattern for audit</li>
```

```
  <li><code>POST /v1/patterns/rollback</code> — Rollback pattern to previous version</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure pattern detection thresholds (minimum sample size, confidence threshold, decay rate).</li>
  <li>Query patterns during context assembly to include learned correlations in reasoning context.</li>
  <li>Use pattern confidence scores to weight recommendations (high-confidence patterns influence NBA more strongly).</li>
  <li>Monitor pattern evolution metrics to detect drift (patterns becoming less predictive over time).</li>
  <li>Freeze patterns during incident response to prevent memory updates from invalidating forensic analysis.</li>
  <li>Review pattern manifests periodically to validate learned correlations and identify spurious patterns.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="340" viewBox="0 0 760 340" xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="320" fill="#FFFFFF" stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-anchor="middle" font-weight="700">Pattern Detection & Learning</text>

    <!-- Input: Impressions -->
    <rect x="40" y="60" width="160" height="80" fill="#F3F4F6" stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="120" y="85" font-size="10" fill="#374151" text-anchor="middle" font-weight="600">Impressions</text>
    <text x="120" y="102" font-size="8" fill="#6B7280" text-anchor="middle">Event sequences</text>
    <text x="120" y="115" font-size="8" fill="#6B7280" text-anchor="middle">Outcome labels</text>
    <text x="120" y="128" font-size="8" fill="#6B7280" text-anchor="middle">Entity co-occurrence</text>

    <!-- Arrow -->
```

```
    <line x1="200" y1="100" x2="240" y2="100" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="240,100 235,97 235,103" fill="#6B7280"/>

    <!-- Detection Process -->
    <rect x="240" y="60" width="280" height="220" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="380" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Detection Process</text>

    <rect x="260" y="100" width="240" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="120" font-size="9" fill="#1E40AF" text-
anchor="middle">Analyze co-occurrence & sequences</text>

    <rect x="260" y="145" width="240" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="165" font-size="9" fill="#1E40AF" text-
anchor="middle">Correlate situations with outcomes</text>

    <rect x="260" y="190" width="240" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="210" font-size="9" fill="#1E40AF" text-
anchor="middle">Apply statistical thresholds</text>

    <rect x="260" y="235" width="240" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="255" font-size="9" fill="#1E40AF" text-
anchor="middle">Update pattern confidence</text>

    <!-- Arrow -->
    <line x1="520" y1="170" x2="560" y2="170" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="560,170 555,167 555,173" fill="#6B7280"/>

    <!-- Output: Patterns -->
    <rect x="560" y="60" width="160" height="220" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="4"/>
    <text x="640" y="85" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">Detected Patterns</text>

    <rect x="575" y="100" width="130" height="50" fill="#FFF1F2"
```

```
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="640" y="118" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Pattern A</text>
    <text x="640" y="131" font-size="7" fill="#991B1B" text-
anchor="middle">Config change → Incident</text>
    <text x="640" y="143" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.85</text>

    <rect x="575" y="160" width="130" height="50" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="640" y="178" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Pattern B</text>
    <text x="640" y="191" font-size="7" fill="#991B1B" text-
anchor="middle">Rollback → Resolution</text>
    <text x="640" y="203" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.78</text>

    <rect x="575" y="220" width="130" height="50" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="640" y="238" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Pattern C</text>
    <text x="640" y="251" font-size="7" fill="#991B1B" text-
anchor="middle">Feature X → Churn risk</text>
    <text x="640" y="263" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.72</text>

    <!-- Learning Loop -->
    <rect x="40" y="290" width="680" height="30" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="380" y="310" font-size="9" fill="#374151" text-
anchor="middle" font-weight="600">Patterns reinforce with
consistent observations | Decay if contradicted | Min sample: 10,
Min confidence: 0.7</text>
  </svg>
</div>

<h3 id="nba-engine">2.7 Next Best Action Engine</h3>

<p><strong>What it does:</strong> The Next Best Action (NBA)
Engine recommends actions based on assembled context, detected
patterns, and outcome predictions. It answers the question:
"Given what we know, what should happen next?" by ranking
```

```
candidate actions using pattern confidence, outcome
probabilities, and policy constraints. Recommendations include
action metadata (expected outcome, confidence score, supporting
evidence) and can operate in advisory mode (suggest only) or
gated execution mode (recommend and execute with approval).</p>

<p><strong>Inputs:</strong> Context manifest from Context
Assembly Engine, detected patterns from Pattern Detection,
candidate actions (pre-defined action catalog or dynamically
generated), and tenant-specific NBA policies (risk tolerance,
approval requirements, action constraints).</p>

<p><strong>Outputs:</strong> Ranked action recommendations with
confidence scores, expected outcome probabilities, supporting
evidence (patterns and impressions that justify recommendation),
and execution metadata (approval status, execution constraints,
rollback procedures).</p>

<p><strong>Persistence model:</strong> NBA recommendations are
logged for audit with full provenance (input context, patterns
used, ranking logic, confidence scores). Executed actions are
tracked with outcome labels for reinforcement learning. NBA
policies (tenant-specific rules for action ranking, approval
thresholds, risk constraints) are stored in tenant configuration.
Recommendation history enables pattern analysis of NBA
effectiveness over time.</p>

<p><strong>Operational behavior:</strong> NBA operates
synchronously with sub-300ms latency target. The engine retrieves
candidate actions from action catalog, scores each action using
pattern confidence and outcome predictions, applies policy
constraints (risk filters, approval requirements), ranks actions
by expected value, and returns top-k recommendations. NBA
supports simulation mode (preview recommendations without
execution) and shadow mode (recommend but do not execute, compare
with actual human decisions). Failures degrade gracefully—if
pattern queries timeout, NBA falls back to rule-based ranking.
</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/nba/recommend</code> — Get action
```

```
recommendations for situation</li>
  <li><code>POST /v1/nba/simulate</code> — Simulate
recommendations without logging</li>
  <li><code>GET /v1/nba/policies</code> — Retrieve tenant NBA
policies</li>
  <li><code>PUT /v1/nba/policies</code> — Update tenant NBA
policies</li>
  <li><code>GET /v1/nba/history</code> — Retrieve recommendation
history</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Define action catalog (available actions with parameters,
constraints, rollback procedures).</li>
  <li>Configure NBA policies per tenant (risk tolerance, approval
thresholds, action constraints).</li>
  <li>Call NBA recommend after context assembly to get ranked
action suggestions.</li>
  <li>Present recommendations to users (advisory mode) or execute
automatically (gated mode with approval).</li>
  <li>Capture outcome labels after action execution to reinforce
patterns and improve future recommendations.</li>
  <li>Use simulation mode during development to test NBA behavior
without affecting production.</li>
  <li>Monitor NBA metrics (recommendation latency, acceptance
rate, outcome accuracy) via observability dashboards.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="380" viewBox="0 0 760 380"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="360" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">Next Best Action Engine</text>

    <!-- Inputs -->
    <rect x="40" y="60" width="140" height="100" fill="#F3F4F6"
stroke="#6B7280" stroke-width="1.5" rx="4"/>
    <text x="110" y="80" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">Inputs</text>
```

```
    <text x="110" y="98" font-size="8" fill="#6B7280" text-
anchor="middle">Context manifest</text>
    <text x="110" y="111" font-size="8" fill="#6B7280" text-
anchor="middle">Detected patterns</text>
    <text x="110" y="124" font-size="8" fill="#6B7280" text-
anchor="middle">Candidate actions</text>
    <text x="110" y="137" font-size="8" fill="#6B7280" text-
anchor="middle">NBA policies</text>

    <!-- Arrow -->
    <line x1="180" y1="110" x2="220" y2="110" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="220,110 215,107 215,113" fill="#6B7280"/>

    <!-- Ranking Process -->
    <rect x="220" y="60" width="320" height="260" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="380" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Ranking Process</text>

    <rect x="240" y="100" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="120" font-size="9" fill="#1E40AF" text-
anchor="middle">1. Retrieve candidate actions from catalog</text>

    <rect x="240" y="145" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="165" font-size="9" fill="#1E40AF" text-
anchor="middle">2. Score actions using pattern confidence</text>

    <rect x="240" y="190" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="210" font-size="9" fill="#1E40AF" text-
anchor="middle">3. Apply policy constraints (risk, approval)
</text>

    <rect x="240" y="235" width="280" height="35" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="255" font-size="9" fill="#1E40AF" text-
anchor="middle">4. Rank by expected value & return top-k</text>

    <rect x="240" y="280" width="280" height="30" fill="#FEF2F2"
```

```
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="380" y="298" font-size="8" fill="#DC2626" text-
anchor="middle" font-weight="600">Modes: Advisory | Gated
Execution | Simulation | Shadow</text>

    <!-- Arrow -->
    <line x1="540" y1="190" x2="580" y2="190" stroke="#6B7280"
stroke-width="1.5"/>
    <polygon points="580,190 575,187 575,193" fill="#6B7280"/>

    <!-- Output: Recommendations -->
    <rect x="580" y="60" width="140" height="260" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="4"/>
    <text x="650" y="85" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">Recommendations</text>

    <rect x="595" y="100" width="110" height="60" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="650" y="118" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Action 1: Rollback</text>
    <text x="650" y="131" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.88</text>
    <text x="650" y="143" font-size="7" fill="#991B1B" text-
anchor="middle">Expected: Resolution</text>
    <text x="650" y="155" font-size="7" fill="#991B1B" text-
anchor="middle">Evidence: Pattern B</text>

    <rect x="595" y="170" width="110" height="60" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="650" y="188" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Action 2: Escalate</text>
    <text x="650" y="201" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.72</text>
    <text x="650" y="213" font-size="7" fill="#991B1B" text-
anchor="middle">Expected: Human review</text>
    <text x="650" y="225" font-size="7" fill="#991B1B" text-
anchor="middle">Evidence: Pattern A</text>

    <rect x="595" y="240" width="110" height="60" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="650" y="258" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Action 3: Monitor</text>
```

```
    <text x="650" y="271" font-size="7" fill="#991B1B" text-
anchor="middle">Confidence: 0.65</text>
    <text x="650" y="283" font-size="7" fill="#991B1B" text-
anchor="middle">Expected: Wait & observe</text>
    <text x="650" y="295" font-size="7" fill="#991B1B" text-
anchor="middle">Evidence: Low risk</text>

    <!-- Target Latency -->
    <rect x="40" y="330" width="680" height="30" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="380" y="350" font-size="9" fill="#374151" text-
anchor="middle" font-weight="600">Target Latency: &lt; 300ms |
Logged for audit | Outcome labels drive reinforcement
learning</text>
  </svg>
</div>

<h3 id="simulation-shadow">2.8 Simulation & Shadow Mode</h3>

<p><strong>What it does:</strong> Simulation & Shadow Mode
enables safe testing and validation of memory-driven
recommendations before production deployment. Simulation mode
allows developers to test NBA recommendations against historical
scenarios without affecting live systems. Shadow mode runs NBA in
parallel with production workflows, comparing AI recommendations
with actual human decisions to measure accuracy, identify gaps,
and build confidence before enabling autonomous execution. Both
modes generate detailed comparison reports for validation and
tuning.</p>

<p><strong>Inputs:</strong> Historical event logs (for
simulation), live production events (for shadow mode), human
decision labels (actual actions taken), and simulation/shadow
policies (comparison metrics, reporting thresholds, validation
criteria).</p>

<p><strong>Outputs:</strong> Simulation reports (NBA
recommendations vs expected outcomes for historical scenarios),
shadow mode reports (NBA recommendations vs actual human
decisions with agreement rates, disagreement analysis, and
confidence distributions), and validation metrics (precision,
recall, F1 score for recommendation accuracy).</p>
```

```
<p><strong>Persistence model:</strong> Simulation runs and shadow mode sessions are logged with full provenance (input scenarios, NBA recommendations, actual outcomes, comparison metrics). Reports are retained for audit and compliance. Simulation scenarios can be saved as test suites for regression testing. Shadow mode data is anonymized for privacy and stored with tenant isolation.</p>

<p><strong>Operational behavior:</strong> Simulation runs asynchronously as batch jobs, processing historical scenarios and generating reports. Shadow mode runs synchronously in production, observing live events and generating NBA recommendations without executing them. Shadow recommendations are logged alongside actual human decisions for comparison. Both modes support configurable comparison metrics (exact match, semantic similarity, outcome equivalence). Simulation and shadow mode do not affect production memory state—recommendations are generated but not reinforced.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/simulation/run</code> — Run simulation on historical scenarios</li>
  <li><code>GET /v1/simulation/reports</code> — Retrieve simulation reports</li>
  <li><code>POST /v1/shadow/enable</code> — Enable shadow mode for tenant</li>
  <li><code>POST /v1/shadow/disable</code> — Disable shadow mode</li>
  <li><code>GET /v1/shadow/reports</code> — Retrieve shadow mode comparison reports</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Create simulation test suites from historical production scenarios (incidents, support tickets, operational events).</li>
  <li>Run simulations to validate NBA recommendations against known outcomes before deploying new patterns or policies.</li>
  <li>Enable shadow mode in production to observe NBA behavior alongside human decisions without risk.</li>
```

```
  <li>Review shadow mode reports to measure agreement rates and identify scenarios where NBA disagrees with human judgment.</li>
  <li>Tune NBA policies and patterns based on simulation and shadow mode feedback to improve accuracy.</li>
  <li>Use shadow mode as a gating criterion for autonomous execution—require 90%+ agreement rate before enabling gated mode.</li>
  <li>Monitor simulation and shadow metrics (agreement rate, confidence distribution, disagreement patterns) via observability dashboards.</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="360" viewBox="0 0 760 360" xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="340" fill="#FFFFFF" stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-anchor="middle" font-weight="700">Simulation & Shadow Mode</text>

    <!-- Simulation Mode -->
    <rect x="40" y="60" width="320" height="130" fill="#F0FDF4" stroke="#10B981" stroke-width="2" rx="6"/>
    <text x="200" y="85" font-size="12" fill="#065F46" text-anchor="middle" font-weight="700">Simulation Mode</text>

    <rect x="60" y="100" width="120" height="70" fill="#DCFCE7" stroke="#10B981" stroke-width="1" rx="3"/>
    <text x="120" y="120" font-size="9" fill="#065F46" text-anchor="middle" font-weight="600">Historical Scenarios</text>
    <text x="120" y="135" font-size="8" fill="#10B981" text-anchor="middle">Past incidents</text>
    <text x="120" y="148" font-size="8" fill="#10B981" text-anchor="middle">Support tickets</text>
    <text x="120" y="161" font-size="8" fill="#10B981" text-anchor="middle">Known outcomes</text>

    <line x1="180" y1="135" x2="200" y2="135" stroke="#10B981" stroke-width="1.5"/>
    <polygon points="200,135 195,132 195,138" fill="#10B981"/>

    <rect x="200" y="100" width="140" height="70" fill="#DCFCE7"
```

```
stroke="#10B981" stroke-width="1" rx="3"/>
    <text x="270" y="120" font-size="9" fill="#065F46" text-
anchor="middle" font-weight="600">NBA Recommendations</text>
    <text x="270" y="135" font-size="8" fill="#10B981" text-
anchor="middle">vs Expected outcomes</text>
    <text x="270" y="150" font-size="8" fill="#10B981" text-
anchor="middle">Accuracy metrics</text>
    <text x="270" y="163" font-size="8" fill="#10B981" text-
anchor="middle">No production impact</text>

    <!-- Shadow Mode -->
    <rect x="400" y="60" width="320" height="130" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="560" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Shadow Mode</text>

    <rect x="420" y="100" width="120" height="70" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="480" y="120" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">Live Production</text>
    <text x="480" y="135" font-size="8" fill="#3B82F6" text-
anchor="middle">Real-time events</text>
    <text x="480" y="148" font-size="8" fill="#3B82F6" text-
anchor="middle">Human decisions</text>
    <text x="480" y="161" font-size="8" fill="#3B82F6" text-
anchor="middle">Actual outcomes</text>

    <line x1="540" y1="135" x2="560" y2="135" stroke="#3B82F6"
stroke-width="1.5"/>
    <polygon points="560,135 555,132 555,138" fill="#3B82F6"/>

    <rect x="560" y="100" width="140" height="70" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="630" y="120" font-size="9" fill="#1E40AF" text-
anchor="middle" font-weight="600">NBA Recommendations</text>
    <text x="630" y="135" font-size="8" fill="#3B82F6" text-
anchor="middle">vs Human decisions</text>
    <text x="630" y="150" font-size="8" fill="#3B82F6" text-
anchor="middle">Agreement rate</text>
    <text x="630" y="163" font-size="8" fill="#3B82F6" text-
anchor="middle">Observe only</text>
```

```
    <!-- Comparison Reports -->
    <rect x="40" y="210" width="680" height="100" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="380" y="235" font-size="12" fill="#DC2626" text-
anchor="middle" font-weight="700">Comparison Reports</text>

    <rect x="60" y="250" width="200" height="45" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="160" y="268" font-size="9" fill="#991B1B" text-
anchor="middle" font-weight="600">Agreement Rate: 87%</text>
    <text x="160" y="283" font-size="8" fill="#991B1B" text-
anchor="middle">High confidence matches: 92%</text>

    <rect x="280" y="250" width="200" height="45" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="380" y="268" font-size="9" fill="#991B1B" text-
anchor="middle" font-weight="600">Disagreement Analysis</text>
    <text x="380" y="283" font-size="8" fill="#991B1B" text-
anchor="middle">NBA more conservative: 8%</text>

    <rect x="500" y="250" width="200" height="45" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="600" y="268" font-size="9" fill="#991B1B" text-
anchor="middle" font-weight="600">Validation Metrics</text>
    <text x="600" y="283" font-size="8" fill="#991B1B" text-
anchor="middle">Precision: 0.89 | Recall: 0.85</text>

    <!-- Deployment Gate -->
    <rect x="40" y="320" width="680" height="30" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="380" y="340" font-size="9" fill="#374151" text-
anchor="middle" font-weight="600">Deployment Gate: Require 90%+
shadow agreement before enabling autonomous execution</text>
  </svg>
</div>

<h3 id="governance-safety">2.9 Governance, Controls & Safety</h3>

<p><strong>What it does:</strong> The Governance, Controls &
Safety component enforces organizational policies, regulatory
compliance, and safety constraints across the memory and action
lifecycle. It provides mechanisms for memory retention policies,
```

```
data decay and deletion, action approval workflows, escalation
rules, audit trails, and emergency controls (freeze, rollback,
circuit breakers). This component ensures that memory-driven AI
operates within acceptable risk boundaries and maintains
compliance with enterprise governance requirements.</p>

<p><strong>Inputs:</strong> Governance policies (retention,
decay, approval thresholds, escalation rules), compliance
requirements (GDPR, SOC2, industry-specific regulations), risk
classifications (action risk levels, memory sensitivity labels),
and audit queries (compliance reports, forensic investigations).
</p>

<p><strong>Outputs:</strong> Policy enforcement decisions
(approve, deny, escalate), audit logs (immutable records of all
memory operations and action executions), compliance reports
(retention compliance, access logs, data lineage), and emergency
control confirmations (freeze acknowledgments, rollback results,
circuit breaker status).</p>

<p><strong>Persistence model:</strong> Governance policies are
versioned and stored in tenant configuration with effective dates
and change history. Audit logs are append-only and immutable,
stored in compliance-grade storage with tamper detection.
Retention metadata is attached to every impression and pattern,
triggering automated deletion when retention periods expire.
Emergency control states (freeze, rollback checkpoints) are
persisted for disaster recovery.</p>

<p><strong>Operational behavior:</strong> Policy enforcement runs
synchronously at decision points (memory write, pattern update,
action execution). Retention enforcement runs asynchronously as
scheduled jobs, scanning for expired memories and executing
deletion. Audit logging is asynchronous with guaranteed delivery—
logs are buffered and retried on failure. Emergency controls
(freeze, rollback) are synchronous and prioritized, blocking all
writes until released. Compliance reports are generated on-demand
or scheduled (monthly, quarterly). The system supports multi-
level approval workflows with escalation paths and timeout
policies.</p>

<p><strong>API touchpoints:</strong></p>
```

```
<ul>
  <li><code>GET /v1/governance/policies</code> — Retrieve
governance policies</li>
  <li><code>PUT /v1/governance/policies</code> — Update
governance policies</li>
  <li><code>POST /v1/governance/freeze</code> — Freeze memory
updates (emergency control)</li>
  <li><code>POST /v1/governance/unfreeze</code> — Release
freeze</li>
  <li><code>POST /v1/governance/rollback</code> — Rollback to
checkpoint</li>
  <li><code>GET /v1/audit/logs</code> — Query audit logs</li>
  <li><code>GET /v1/audit/compliance</code> — Generate compliance
report</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
  <li>Configure governance policies per tenant (retention: 2
years, decay: 30-day half-life, approval threshold: high-risk
actions).</li>
  <li>Define action risk classifications (low, medium, high) with
corresponding approval workflows.</li>
  <li>Implement escalation rules (timeout after 1 hour → escalate
to manager, timeout after 4 hours → auto-deny).</li>
  <li>Use freeze control during incident response to prevent
memory updates from contaminating forensic analysis.</li>
  <li>Execute rollback to restore memory state to known-good
checkpoint after detecting data corruption or policy violations.
</li>
  <li>Query audit logs for compliance reporting, security
investigations, and operational forensics.</li>
  <li>Monitor governance metrics (policy violations, approval
latency, retention compliance rate) via observability dashboards.
</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="400" viewBox="0 0 760 400"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="380" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
```

```
    <text x="380" y="35" font-size="14" fill="#111827" text-anchor="middle" font-weight="700">Governance, Controls & Safety</text>

    <!-- Policy Enforcement -->
    <rect x="40" y="60" width="210" height="150" fill="#FEF2F2" stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="145" y="85" font-size="11" fill="#DC2626" text-anchor="middle" font-weight="700">Policy Enforcement</text>

    <rect x="60" y="100" width="170" height="30" fill="#FFF1F2" stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="145" y="118" font-size="8" fill="#991B1B" text-anchor="middle">Retention & Decay</text>

    <rect x="60" y="140" width="170" height="30" fill="#FFF1F2" stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="145" y="158" font-size="8" fill="#991B1B" text-anchor="middle">Approval Workflows</text>

    <rect x="60" y="180" width="170" height="30" fill="#FFF1F2" stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="145" y="198" font-size="8" fill="#991B1B" text-anchor="middle">Escalation Rules</text>

    <!-- Audit & Compliance -->
    <rect x="270" y="60" width="210" height="150" fill="#EFF6FF" stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="375" y="85" font-size="11" fill="#1E40AF" text-anchor="middle" font-weight="700">Audit & Compliance</text>

    <rect x="290" y="100" width="170" height="30" fill="#DBEAFE" stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="375" y="118" font-size="8" fill="#1E40AF" text-anchor="middle">Immutable Audit Logs</text>

    <rect x="290" y="140" width="170" height="30" fill="#DBEAFE" stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="375" y="158" font-size="8" fill="#1E40AF" text-anchor="middle">Compliance Reports</text>

    <rect x="290" y="180" width="170" height="30" fill="#DBEAFE"
```

```
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="375" y="198" font-size="8" fill="#1E40AF" text-
anchor="middle">Data Lineage</text>

    <!-- Emergency Controls -->
    <rect x="500" y="60" width="220" height="150" fill="#FEF3C7"
stroke="#F59E0B" stroke-width="2" rx="6"/>
    <text x="610" y="85" font-size="11" fill="#92400E" text-
anchor="middle" font-weight="700">Emergency Controls</text>

    <rect x="520" y="100" width="180" height="30" fill="#FEF9C3"
stroke="#F59E0B" stroke-width="1" rx="3"/>
    <text x="610" y="118" font-size="8" fill="#92400E" text-
anchor="middle" font-weight="600">Freeze (block writes)</text>

    <rect x="520" y="140" width="180" height="30" fill="#FEF9C3"
stroke="#F59E0B" stroke-width="1" rx="3"/>
    <text x="610" y="158" font-size="8" fill="#92400E" text-
anchor="middle" font-weight="600">Rollback (restore checkpoint)
</text>

    <rect x="520" y="180" width="180" height="30" fill="#FEF9C3"
stroke="#F59E0B" stroke-width="1" rx="3"/>
    <text x="610" y="198" font-size="8" fill="#92400E" text-
anchor="middle" font-weight="600">Circuit Breaker</text>

    <!-- Compliance Standards -->
    <rect x="40" y="230" width="680" height="80" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="6"/>
    <text x="380" y="255" font-size="11" fill="#374151" text-
anchor="middle" font-weight="700">Compliance Standards</text>

    <rect x="60" y="270" width="140" height="30" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="1" rx="3"/>
    <text x="130" y="288" font-size="8" fill="#6B7280" text-
anchor="middle">GDPR (data deletion)</text>

    <rect x="220" y="270" width="140" height="30" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="1" rx="3"/>
    <text x="290" y="288" font-size="8" fill="#6B7280" text-
anchor="middle">SOC 2 (audit trails)</text>
```

```
    <rect x="380" y="270" width="140" height="30" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="1" rx="3"/>
    <text x="450" y="288" font-size="8" fill="#6B7280" text-
anchor="middle">HIPAA (data sensitivity)</text>

    <rect x="540" y="270" width="160" height="30" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="1" rx="3"/>
    <text x="620" y="288" font-size="8" fill="#6B7280" text-
anchor="middle">Industry-specific</text>

    <!-- Governance Flow -->
    <rect x="40" y="330" width="680" height="50" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1" rx="4"/>
    <text x="380" y="350" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">Governance Flow</text>
    <text x="380" y="368" font-size="8" fill="#991B1B" text-
anchor="middle">Policy check → Approval (if required) → Audit log
→ Execute → Outcome log → Retention enforcement</text>
  </svg>
</div>

<h3 id="platform-apis">2.10 Platform APIs & Integration
Surface</h3>

<p><strong>What it does:</strong> The Platform APIs & Integration
Surface provides a comprehensive RESTful API layer for external
systems to interact with Fabric Mind. It exposes all platform
capabilities—event ingestion, context assembly, NBA
recommendations, governance controls, and audit queries—through
versioned, documented, and rate-limited endpoints. The
integration surface includes SDKs (Python, JavaScript, Go),
webhooks for event-driven integration, and batch APIs for high-
throughput scenarios. This component ensures that Fabric Mind
integrates seamlessly into existing enterprise architectures.</p>

<p><strong>Inputs:</strong> API requests from external systems
(agent frameworks, workflow orchestrators, observability
platforms, business applications), authentication tokens (service
identity, user SSO), and integration configurations (webhook
URLs, batch schedules, rate limits).</p>

<p><strong>Outputs:</strong> API responses with structured
```

```
payloads (JSON), error messages with actionable guidance, webhook
notifications for asynchronous events, and integration health
metrics (API latency, error rates, quota usage).</p>

<p><strong>Persistence model:</strong> API request logs are
retained for debugging and billing. Integration configurations
(webhook URLs, API keys, rate limits) are stored in tenant
configuration. API usage metrics are aggregated for billing and
capacity planning. Webhook delivery receipts are logged for
reliability tracking.</p>

<p><strong>Operational behavior:</strong> APIs are synchronous
with sub-500ms latency targets (event ingestion, context
assembly, NBA recommend). Batch APIs are asynchronous with job
status tracking. Webhooks are fire-and-forget with retry logic
(exponential backoff, max 3 retries). Rate limiting is enforced
per tenant with configurable quotas (default: 1000
requests/minute). API versioning follows semantic versioning with
backward compatibility guarantees. The integration surface
supports OpenAPI specifications for auto-generated client
libraries.</p>

<p><strong>API touchpoints:</strong></p>
<ul>
  <li><code>POST /v1/events</code> — Ingest events (single or
batch)</li>
  <li><code>POST /v1/context/assemble</code> — Assemble context
for reasoning</li>
  <li><code>POST /v1/nba/recommend</code> — Get action
recommendations</li>
  <li><code>POST /v1/actions/execute</code> — Execute gated
action</li>
  <li><code>GET /v1/audit/logs</code> — Query audit logs</li>
  <li><code>POST /v1/webhooks/register</code> — Register webhook
for events</li>
  <li><code>GET /v1/health</code> — Health check endpoint</li>
  <li><code>GET /v1/metrics</code> — Integration health
metrics</li>
</ul>

<p><strong>How developers use it:</strong></p>
<ol>
```

```
  <li>Authenticate using service identity tokens (OAuth 2.0
client credentials flow) or user SSO (OIDC).</li>
  <li>Install SDK for preferred language (Python, JavaScript, Go)
or use raw REST API with OpenAPI spec.</li>
  <li>Integrate event ingestion into existing observability
pipelines (send logs, metrics, traces to Fabric Mind).</li>
  <li>Call context assembly before invoking reasoning engine to
inject memory-driven context.</li>
  <li>Use NBA recommend to get action suggestions and present to
users or execute autonomously (gated mode).</li>
  <li>Register webhooks to receive notifications for asynchronous
events (pattern detected, approval required, action completed).
</li>
  <li>Monitor API usage metrics (latency, error rate, quota
consumption) via observability dashboards.</li>
  <li>Use batch APIs for high-throughput scenarios (bulk event
ingestion, historical data backfill).</li>
</ol>

<div class="diagram-container">
  <svg width="760" height="420" viewBox="0 0 760 420"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="400" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">Platform APIs & Integration
Surface</text>

    <!-- External Systems -->
    <rect x="40" y="60" width="680" height="80" fill="#F3F4F6"
stroke="#6B7280" stroke-width="2" rx="6"/>
    <text x="380" y="85" font-size="11" fill="#374151" text-
anchor="middle" font-weight="700">External Systems</text>

    <rect x="60" y="100" width="130" height="30" fill="#FFFFFF"
stroke="#6B7280" stroke-width="1" rx="3"/>
    <text x="125" y="118" font-size="8" fill="#6B7280" text-
anchor="middle">Agent Frameworks</text>

    <rect x="210" y="100" width="130" height="30" fill="#FFFFFF"
stroke="#6B7280" stroke-width="1" rx="3"/>
    <text x="275" y="118" font-size="8" fill="#6B7280" text-
```

```
anchor="middle">Workflow Orchestrators</text>

    <rect x="360" y="100" width="130" height="30" fill="#FFFFFF"
stroke="#6B7280" stroke-width="1" rx="3"/>
    <text x="425" y="118" font-size="8" fill="#6B7280" text-
anchor="middle">Observability Platforms</text>

    <rect x="510" y="100" width="130" height="30" fill="#FFFFFF"
stroke="#6B7280" stroke-width="1" rx="3"/>
    <text x="575" y="118" font-size="8" fill="#6B7280" text-
anchor="middle">Business Applications</text>

    <!-- Arrow Down -->
    <line x1="380" y1="140" x2="380" y2="170" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="380,170 377,165 383,165" fill="#6B7280"/>

    <!-- API Layer -->
    <rect x="40" y="170" width="680" height="180" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="380" y="195" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">RESTful API Layer</text>

    <!-- Core APIs -->
    <rect x="60" y="210" width="200" height="130" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="4"/>
    <text x="160" y="230" font-size="10" fill="#1E40AF" text-
anchor="middle" font-weight="600">Core APIs</text>
    <text x="160" y="250" font-size="8" fill="#3B82F6" text-
anchor="middle">POST /v1/events</text>
    <text x="160" y="265" font-size="8" fill="#3B82F6" text-
anchor="middle">POST /v1/context/assemble</text>
    <text x="160" y="280" font-size="8" fill="#3B82F6" text-
anchor="middle">POST /v1/nba/recommend</text>
    <text x="160" y="295" font-size="8" fill="#3B82F6" text-
anchor="middle">POST /v1/actions/execute</text>
    <text x="160" y="310" font-size="8" fill="#3B82F6" text-
anchor="middle">GET /v1/audit/logs</text>
    <text x="160" y="328" font-size="7" fill="#1E40AF" text-
anchor="middle" font-style="italic">Latency: &lt; 500ms</text>

    <!-- Integration Features -->
```

```
    <rect x="280" y="210" width="200" height="130" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="4"/>
    <text x="380" y="230" font-size="10" fill="#1E40AF" text-
anchor="middle" font-weight="600">Integration Features</text>
    <text x="380" y="250" font-size="8" fill="#3B82F6" text-
anchor="middle">SDKs (Python, JS, Go)</text>
    <text x="380" y="265" font-size="8" fill="#3B82F6" text-
anchor="middle">Webhooks (event-driven)</text>
    <text x="380" y="280" font-size="8" fill="#3B82F6" text-
anchor="middle">Batch APIs (high-throughput)</text>
    <text x="380" y="295" font-size="8" fill="#3B82F6" text-
anchor="middle">OpenAPI Spec</text>
    <text x="380" y="310" font-size="8" fill="#3B82F6" text-
anchor="middle">Rate Limiting (1K req/min)</text>
    <text x="380" y="328" font-size="7" fill="#1E40AF" text-
anchor="middle" font-style="italic">Versioned & Documented</text>

    <!-- Auth & Security -->
    <rect x="500" y="210" width="200" height="130" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="4"/>
    <text x="600" y="230" font-size="10" fill="#1E40AF" text-
anchor="middle" font-weight="600">Auth & Security</text>
    <text x="600" y="250" font-size="8" fill="#3B82F6" text-
anchor="middle">OAuth 2.0 (service tokens)</text>
    <text x="600" y="265" font-size="8" fill="#3B82F6" text-
anchor="middle">OIDC (user SSO)</text>
    <text x="600" y="280" font-size="8" fill="#3B82F6" text-
anchor="middle">Tenant isolation</text>
    <text x="600" y="295" font-size="8" fill="#3B82F6" text-
anchor="middle">API key rotation</text>
    <text x="600" y="310" font-size="8" fill="#3B82F6" text-
anchor="middle">TLS 1.3</text>
    <text x="600" y="328" font-size="7" fill="#1E40AF" text-
anchor="middle" font-style="italic">Enterprise-grade
security</text>

    <!-- Arrow Down -->
    <line x1="380" y1="350" x2="380" y2="370" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="380,370 377,365 383,365" fill="#6B7280"/>

    <!-- Fabric Mind Platform -->
```

```
    <rect x="40" y="370" width="680" height="30" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="4"/>
    <text x="380" y="390" font-size="10" fill="#DC2626" text-
anchor="middle" font-weight="600">Fabric Mind Platform (Memory
Store, Context Assembly, NBA, Governance)</text>
  </svg>
</div>

<h2 id="developer-access">3. Developer Access &
Implementation</h2>

<h3 id="auth-identity">Authentication & Identity</h3>

<p>Fabric Mind supports enterprise-grade authentication and
identity management to ensure secure access for both human users
and automated services. The platform integrates with existing
identity providers through standard protocols, enabling seamless
adoption without requiring new credential management
infrastructure.</p>

<p><strong>SSO/OIDC for Humans:</strong> Human users authenticate
via Single Sign-On (SSO) using OpenID Connect (OIDC). Fabric Mind
acts as a relying party, delegating authentication to the
organization's identity provider (Okta, Azure AD, Google
Workspace, etc.). After successful authentication, users receive
JWT tokens with claims (user ID, email, roles, tenant ID) that
govern API access and UI permissions. Token lifetime is
configurable (default: 1 hour with refresh tokens valid for 7
days). Multi-factor authentication (MFA) enforcement is inherited
from the identity provider.</p>

<p><strong>Service Identity Tokens for Services:</strong>
Automated services (agent frameworks, workflow orchestrators,
batch jobs) authenticate using OAuth 2.0 client credentials flow.
Services are issued client IDs and secrets (rotatable API keys)
that are exchanged for short-lived access tokens (default: 15
minutes). Service tokens include tenant ID and permission scopes
(read-only, write, admin) that restrict API access. Token
rotation is automated with grace periods to prevent service
disruption during key updates.</p>

<p><strong>Tenant Isolation:</strong> All API requests are scoped
```

```
to a tenant ID derived from authentication tokens. Tenant
isolation is enforced at every layer: memory queries cannot
access other tenants' data, context assembly respects tenant
boundaries, NBA recommendations are tenant-specific, and audit
logs are segregated by tenant. Multi-tenant deployments use
logical isolation with per-tenant encryption keys. Dedicated
tenant deployments provide physical isolation with separate
infrastructure.</p>

<p><strong>Permission-Aware Retrieval and Action Execution:
</strong> Context assembly and NBA recommendations respect user
permissions. If a user lacks permission to view certain memories
(e.g., sensitive customer data), those impressions are filtered
from context manifests. Similarly, NBA recommendations exclude
actions the user is not authorized to execute. Permission checks
are performed at query time using role-based access control
(RBAC) policies configured per tenant.</p>


<h3 id="integration-patterns">Integration Patterns</h3>

<p>Fabric Mind provides five core integration patterns that cover
the full memory lifecycle: event ingestion, context assembly,
action recommendation, gated execution, and audit querying. Each
pattern includes request/response examples with JSON payloads to
accelerate integration development.</p>

<h4>A) Event Ingestion</h4>

<p>Ingest events from observability pipelines, application logs,
user interactions, or business transactions. Events can be sent
individually or in batches for high-throughput scenarios. Each
event includes a type, timestamp, entities, and optional metadata
for provenance tracking.</p>

<pre><code>POST /v1/events
```

Content-Type: application/json Authorization: Bearer {access_token}

{ “events”: [

```
{
  "event_id": "evt_abc123",
  "event_type": "incident.detected",
  "timestamp": "2026-01-09T14:30:00Z",
  "entities": {
    "service": "payment-api",
    "severity": "high",
    "error_code": "500"
  },
  "context": {
    "deployment_id": "deploy_xyz789",
    "region": "us-west-2"
  },
  "metadata": {
    "source": "datadog",
    "trace_id": "trace_456def"
  }
}
```

] }

Response: { “status”: “accepted”, “ingested_count”: 1, “impression_ids”: [“imp_789ghi”] }

```
<h4>B) Context Assembly</h4>

<p>Assemble relevant context for a reasoning engine by querying
memory with a situation signature. The response includes a
bounded manifest of relevant impressions, graph-connected
memories, confidence scores, and provenance metadata. The
manifest is designed to fit within reasoning token budgets while
maximizing relevance.</p>

<pre><code>POST /v1/context/assemble
```

Content-Type: application/json Authorization: Bearer {access_token}

{ “situation”: {

```
"event_type": "incident.detected",
"entities": {
  "service": "payment-api",
  "severity": "high"
}
```

}, “query_params”: {

```
"similarity_threshold": 0.7,
"temporal_window_days": 30,
"max_results": 20,
"graph_depth": 2
```

} }

Response: { “manifest_id”: “manifest_jkl012”, “impressions”: [

```
{
  "impression_id": "imp_345mno",
  "similarity_score": 0.92,
  "freshness_days": 7,
  "confidence": 0.88,
  "entities": {
    "service": "payment-api",
    "action": "rollback",
    "outcome": "resolved"
  },
  "provenance": {
    "timestamp": "2026-01-02T10:15:00Z",
    "source": "incident_response"
  }
}
```

], “graph_connected”: [

```
{
  "impression_id": "imp_678pqr",
  "relationship": "causal",
  "confidence": 0.85
}
```

], “metadata”: {

```
"result_count": 12,
"truncated": false,
"assembly_latency_ms": 145
```

} }

```
<h4>C) NBA Recommend</h4>

<p>Get ranked action recommendations based on assembled context
and detected patterns. Recommendations include confidence scores,
expected outcomes, and supporting evidence. Use advisory mode to
present suggestions to users or gated mode for autonomous
execution with approval.</p>

<pre><code>POST /v1/nba/recommend
```

Content-Type: application/json Authorization: Bearer {access_token}

{ “manifest_id”: “manifest_jkl012”, “candidate_actions”: [

```
"rollback_deployment",
"escalate_to_oncall",
"restart_service",
"monitor_and_wait"
```

], “mode”: “advisory” }

Response: { “recommendations”: [

```
{
  "action": "rollback_deployment",
  "confidence": 0.88,
  "expected_outcome": "incident_resolved",
  "outcome_probability": 0.82,
  "supporting_evidence": [
    {
      "pattern_id": "pattern_stu901",
      "description": "Rollback resolves payment-api incidents 82%
of the time",
      "confidence": 0.85
    }
  ],
  "execution_metadata": {
    "approval_required": false,
    "risk_level": "medium",
    "rollback_procedure": "revert_to_previous_version"
  }
},
{
  "action": "escalate_to_oncall",
  "confidence": 0.72,
  "expected_outcome": "human_review",
  "outcome_probability": 0.95,
  "supporting_evidence": [
    {
      "pattern_id": "pattern_vwx234",
      "description": "High-severity incidents benefit from human
review",
      "confidence": 0.78
    }
  ],
  "execution_metadata": {
    "approval_required": true,
    "risk_level": "low"
  }
}
```


]}

```
<h4>D) Gated Execute</h4>

<p>Execute an action with approval workflow. Gated execution
requires explicit approval for high-risk actions, logs the
execution for audit, and captures outcome labels for
reinforcement learning. Execution can be synchronous (wait for
completion) or asynchronous (return job ID).</p>

<pre><code>POST /v1/actions/execute
```

Content-Type: application/json Authorization: Bearer {access_token}

{ “action”: “rollback_deployment”, “parameters”: {

```
"service": "payment-api",
"target_version": "v1.2.3"
```

}, “approval”: {

```
"approved_by": "user@example.com",
"approval_timestamp": "2026-01-09T14:35:00Z"
```

}, “execution_mode”: “synchronous” }

Response: { “execution_id”: “exec_yza567”, “status”: “completed”, “outcome”: “success”, “outcome_label”: “incident_resolved”, “execution_log”: {

```
"started_at": "2026-01-09T14:35:05Z",
"completed_at": "2026-01-09T14:37:12Z",
"duration_seconds": 127
```

}, “audit_trail_id”: “audit_bcd890” }

```
<h4>E) Audit Query</h4>

<p>Query audit logs for compliance reporting, security
investigations, or operational forensics. Audit logs are
immutable and include full provenance for all memory operations
and action executions. Queries support filtering by time range,
event type, user, and tenant.</p>

<pre><code>GET /v1/audit/logs?start_time=2026-01-
01T00:00:00Z&end_time=2026-01-
09T23:59:59Z&event_type=action.executed&limit=100
```

Authorization: Bearer {access_token}

Response: { “logs”: [

```
{
  "audit_id": "audit_bcd890",
  "timestamp": "2026-01-09T14:35:05Z",
  "event_type": "action.executed",
  "user_id": "user@example.com",
  "tenant_id": "tenant_123",
  "action": "rollback_deployment",
  "outcome": "success",
  "metadata": {
    "execution_id": "exec_yza567",
    "approval_required": false,
    "risk_level": "medium"
  }
}
```

], “pagination”: {

```
  "total_count": 1,
  "next_cursor": null
```


}}

```
<h3 id="implementation-playbooks">Implementation Playbooks</h3>

<p>The following playbooks provide step-by-step implementation guidance for three common agentic AI use cases: Customer Support, Incident/Ops, and Coding/DevEx agents. Each playbook covers event sources, outcomes to capture, context assembly integration, safe rollout strategy, and ROI measurement.</p>

<h4>Playbook 1: Customer Support Agent</h4>

<p><strong>Event Sources:</strong> Support ticket creation (Zendesk, Intercom, Salesforce Service Cloud), customer interactions (chat transcripts, email threads), resolution actions (ticket closed, escalated, reassigned), and customer feedback (CSAT scores, follow-up tickets).</p>

<p><strong>Outcomes to Capture:</strong> Resolution success (ticket resolved without escalation), resolution time (time to first response, time to resolution), customer satisfaction (CSAT score, NPS), and escalation rate (percentage of tickets escalated to human agents).</p>

<p><strong>Context Assembly Integration:</strong> Before generating a response, call <code>POST /v1/context/assemble</code> with the current ticket signature (customer ID, issue category, product area). The context manifest includes similar past tickets, successful resolution patterns, and customer history (previous issues, preferences, sentiment). Inject this context into the reasoning prompt to ground responses in organizational memory.</p>

<p><strong>Safe Rollout Strategy:</strong></p>
<ol>
  <li><strong>Shadow Mode (Week 1-2):</strong> Enable shadow mode to observe NBA recommendations alongside human agent decisions. Measure agreement rate (target: 80%+). Identify scenarios where NBA disagrees with human judgment and tune patterns accordingly.</li>
  <li><strong>Gated Mode - Low Risk (Week 3-4):</strong> Enable gated execution for low-risk actions (send knowledge base article, request additional information). Require human approval
```

```
for medium/high-risk actions (issue refund, escalate to
engineering). Monitor outcome accuracy and customer satisfaction.
</li>
  <li><strong>Expand Autonomy (Week 5-8):</strong> Gradually
expand autonomous execution to medium-risk actions as confidence
grows (agreement rate > 90%, CSAT maintained or improved).
Maintain human-in-the-loop for high-risk actions (account
changes, policy exceptions).</li>
  <li><strong>Full Production (Week 9+):</strong> Operate in
advisory mode for complex cases and autonomous mode for routine
cases. Continuously monitor ROI metrics and tune patterns based
on outcome feedback.</li>
</ol>

<p><strong>ROI Measurement:</strong> Track Average Handle Time
(AHT) reduction (target: 20-30% decrease), First Contact
Resolution (FCR) improvement (target: 10-15% increase),
escalation rate reduction (target: 15-25% decrease), and CSAT
maintenance or improvement (target: maintain baseline or +5%
improvement). Calculate cost savings based on AHT reduction
multiplied by agent hourly cost and ticket volume.</p>

<h4>Playbook 2: Incident/Ops Agent</h4>

<p><strong>Event Sources:</strong> Incident detection (PagerDuty,
Datadog, New Relic), deployment events (CI/CD pipelines,
Kubernetes rollouts), configuration changes (Terraform, Ansible),
resolution actions (rollback, restart, scale-up), and post-
incident reviews (RCA documents, action items).</p>

<p><strong>Outcomes to Capture:</strong> Incident resolution
success (incident resolved without escalation), Mean Time To
Resolution (MTTR), false positive rate (alerts that did not
require action), and action effectiveness (percentage of
recommended actions that resolved incidents).</p>

<p><strong>Context Assembly Integration:</strong> When an
incident is detected, call <code>POST /v1/context/assemble</code>
with the incident signature (service, error code, severity). The
context manifest includes similar past incidents, successful
resolution patterns (e.g., "rollback resolves payment-api 500
errors 85% of the time"), and causal relationships (e.g., "config
```

```
changes on Fridays correlate with incidents"). Use this context
to prioritize NBA recommendations and accelerate triage.</p>

<p><strong>Safe Rollout Strategy:</strong></p>
<ol>
  <li><strong>Shadow Mode (Week 1-3):</strong> Run NBA in shadow
mode during incident response. Compare NBA recommendations with
actual on-call engineer decisions. Measure agreement rate
(target: 75%+) and identify gaps in pattern detection or context
assembly.</li>
  <li><strong>Gated Mode - Low Risk (Week 4-6):</strong> Enable
gated execution for low-risk actions (collect diagnostic logs,
restart non-critical services). Require approval for medium/high-
risk actions (rollback production deployments, scale down
services). Monitor MTTR and false positive rate.</li>
  <li><strong>Expand Autonomy (Week 7-10):</strong> Expand
autonomous execution to medium-risk actions as confidence grows
(agreement rate > 85%, MTTR improved). Maintain human approval
for high-risk actions (database rollbacks, cross-region
failovers).</li>
  <li><strong>Full Production (Week 11+):</strong> Operate in
autonomous mode for routine incidents (known patterns with high
confidence) and advisory mode for novel incidents. Continuously
refine patterns based on post-incident reviews.</li>
</ol>

<p><strong>ROI Measurement:</strong> Track MTTR reduction
(target: 30-40% decrease), on-call engineer toil reduction
(target: 25-35% decrease in manual interventions), false positive
rate reduction (target: 20-30% decrease), and incident recurrence
rate (target: 15-20% decrease). Calculate cost savings based on
reduced on-call hours and prevented downtime costs.</p>

<h4>Playbook 3: Coding/DevEx Agent</h4>

<p><strong>Event Sources:</strong> Code review requests (GitHub,
GitLab pull requests), build failures (CI/CD pipelines),
developer questions (Slack, internal forums), code changes
(commits, diffs), and resolution actions (code suggestions,
documentation links, automated fixes).</p>

<p><strong>Outcomes to Capture:</strong> Code review quality
```

```
(suggestions accepted by developers), build fix success (build
passes after applying suggestion), developer satisfaction
(feedback on suggestion quality), and time to resolution (time
from question to answer, time from build failure to fix).</p>

<p><strong>Context Assembly Integration:</strong> When a
developer asks a question or a build fails, call <code>POST
/v1/context/assemble</code> with the query signature (code
context, error message, repository). The context manifest
includes similar past issues, successful resolution patterns
(e.g., "dependency version mismatches resolved by updating
package.json"), and relevant documentation. Use this context to
generate grounded code suggestions and accelerate developer
workflows.</p>

<p><strong>Safe Rollout Strategy:</strong></p>
<ol>
  <li><strong>Shadow Mode (Week 1-2):</strong> Run NBA in shadow
mode for code review and build failures. Compare NBA suggestions
with actual developer actions. Measure agreement rate (target:
70%+) and identify areas where suggestions are off-target.</li>
  <li><strong>Advisory Mode (Week 3-6):</strong> Present NBA
suggestions to developers as recommendations (not automated
fixes). Track acceptance rate (target: 60%+) and gather developer
feedback to tune patterns and improve suggestion quality.</li>
  <li><strong>Gated Automation (Week 7-10):</strong> Enable
automated fixes for low-risk issues (formatting, linting,
dependency updates) with developer review. Require manual
approval for medium/high-risk changes (logic changes, API
modifications). Monitor build success rate and developer
satisfaction.</li>
  <li><strong>Full Production (Week 11+):</strong> Operate in
autonomous mode for routine fixes and advisory mode for complex
changes. Continuously refine patterns based on developer feedback
and code review outcomes.</li>
</ol>

<p><strong>ROI Measurement:</strong> Track developer productivity
improvement (target: 15-25% reduction in time spent on routine
tasks), build fix time reduction (target: 40-50% decrease in time
to fix build failures), code review cycle time reduction (target:
20-30% decrease), and developer satisfaction (target: maintain
```

```
baseline or +10% improvement). Calculate cost savings based on
developer time saved multiplied by developer hourly cost.</p>


<h3 id="integration-loop">Integration Loop Diagram</h3>

<p>The following diagram illustrates the end-to-end integration
loop for memory-driven agentic AI, showing how events flow
through Fabric Mind, how context is assembled for reasoning, how
actions are recommended and executed, and how outcomes feed back
into memory for continuous learning.</p>

<div class="diagram-container">
  <svg width="760" height="480" viewBox="0 0 760 480"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="460" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">End-to-End Integration
Loop</text>

    <!-- External System -->
    <rect x="40" y="60" width="140" height="60" fill="#F3F4F6"
stroke="#6B7280" stroke-width="2" rx="4"/>
    <text x="110" y="85" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">External System</text>
    <text x="110" y="100" font-size="8" fill="#6B7280" text-
anchor="middle">Agent / Workflow</text>

    <!-- Arrow: Event Ingestion -->
    <line x1="180" y1="90" x2="220" y2="90" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="220,90 215,87 215,93" fill="#6B7280"/>
    <text x="200" y="85" font-size="8" fill="#6B7280" text-
anchor="middle">1. Ingest</text>

    <!-- Fabric Mind: Event Processing -->
    <rect x="220" y="60" width="320" height="360" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="380" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Fabric Mind Platform</text>
```

```
    <!-- Step 1: Impression Engine -->
    <rect x="240" y="100" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="123" font-size="9" fill="#1E40AF" text-
anchor="middle">Impression Engine → Memory Store</text>

    <!-- Step 2: Context Assembly -->
    <rect x="240" y="150" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="173" font-size="9" fill="#1E40AF" text-
anchor="middle">Context Assembly (query memory + graph)</text>

    <!-- Step 3: Pattern Detection -->
    <rect x="240" y="200" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="223" font-size="9" fill="#1E40AF" text-
anchor="middle">Pattern Detection (correlations + trends)</text>

    <!-- Step 4: NBA Recommend -->
    <rect x="240" y="250" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="273" font-size="9" fill="#1E40AF" text-
anchor="middle">NBA Recommend (rank actions)</text>

    <!-- Step 5: Governance Check -->
    <rect x="240" y="300" width="280" height="40" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="380" y="323" font-size="9" fill="#DC2626" text-
anchor="middle" font-weight="600">Governance Check (approval if
required)</text>

    <!-- Step 6: Action Execute -->
    <rect x="240" y="350" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="380" y="373" font-size="9" fill="#1E40AF" text-
anchor="middle">Action Execute (gated or advisory)</text>

    <!-- Arrow: Context + Recommendations -->
    <line x1="540" y1="220" x2="580" y2="220" stroke="#6B7280"
stroke-width="2"/>
    <polygon points="580,220 575,217 575,223" fill="#6B7280"/>
    <text x="560" y="215" font-size="8" fill="#6B7280" text-
```

```
anchor="middle">2. Context</text>
    <text x="560" y="227" font-size="8" fill="#6B7280" text-
anchor="middle">+ NBA</text>

    <!-- External System: Reasoning + Action -->
    <rect x="580" y="180" width="140" height="80" fill="#F3F4F6"
stroke="#6B7280" stroke-width="2" rx="4"/>
    <text x="650" y="205" font-size="10" fill="#374151" text-
anchor="middle" font-weight="600">Reasoning Engine</text>
    <text x="650" y="220" font-size="8" fill="#6B7280" text-
anchor="middle">+ Action Execution</text>
    <text x="650" y="235" font-size="8" fill="#6B7280" text-
anchor="middle">(with memory context)</text>

    <!-- Arrow: Outcome Feedback -->
    <line x1="650" y1="260" x2="650" y2="300" stroke="#10B981"
stroke-width="2"/>
    <line x1="650" y1="300" x2="540" y2="300" stroke="#10B981"
stroke-width="2"/>
    <polygon points="540,300 545,297 545,303" fill="#10B981"/>
    <text x="595" y="295" font-size="8" fill="#10B981" text-
anchor="middle" font-weight="600">3. Outcome</text>

    <!-- Learning Loop -->
    <rect x="40" y="440" width="680" height="20" fill="#F0FDF4"
stroke="#10B981" stroke-width="1" rx="4"/>
    <text x="380" y="454" font-size="8" fill="#065F46" text-
anchor="middle" font-weight="600">Continuous Learning Loop:
Outcomes reinforce patterns → Improve future
recommendations</text>
  </svg>
</div>


<h2 id="enterprise-readiness">4. Enterprise Readiness</h2>

<h3 id="deployment-modes">Deployment Modes</h3>

<p>Fabric Mind supports three deployment modes to meet diverse
enterprise requirements for control, compliance, and cost
optimization.</p>
```

```
<p><strong>Managed SaaS:</strong> Fully managed multi-tenant deployment hosted by Fabric Mind. Tenants share infrastructure with logical isolation (per-tenant encryption keys, network segmentation). This mode offers the fastest time to value with zero infrastructure management overhead. Suitable for organizations prioritizing speed and simplicity over infrastructure control. Data residency options available (US, EU, APAC regions).</p>

<p><strong>Dedicated Tenant:</strong> Single-tenant deployment in Fabric Mind-managed infrastructure. Provides physical isolation with dedicated compute, storage, and network resources. Suitable for organizations with strict compliance requirements (HIPAA, FedRAMP) or performance isolation needs. Includes dedicated support SLA and custom retention policies.</p>

<p><strong>On-Premises (Roadmap):</strong> Self-hosted deployment in customer-managed infrastructure (private cloud, on-premises data centers). Provides maximum control over data sovereignty, network topology, and infrastructure configuration. Requires customer-managed operations (upgrades, backups, monitoring). Suitable for organizations with air-gapped environments or regulatory constraints prohibiting cloud deployment. Expected availability: Q3 2026.</p>

<h3 id="reliability-slos">Reliability and SLOs</h3>

<p>Fabric Mind operates under enterprise-grade Service Level Objectives (SLOs) with transparent monitoring and incident communication.</p>

<p><strong>Availability:</strong> 99.9% uptime for Managed SaaS (43 minutes downtime per month), 99.95% for Dedicated Tenant (22 minutes downtime per month). Measured as percentage of successful API requests over rolling 30-day window. Excludes scheduled maintenance windows (announced 7 days in advance, limited to 4 hours per quarter).</p>

<p><strong>Ingest Latency:</strong> P95 latency < 100ms for event ingestion (from API request to impression write). P99 latency < 200ms. Measured at API gateway, excludes client network latency.
</p>
```

```
<p><strong>Assembly Latency:</strong> P95 latency < 200ms for context assembly (from request to manifest response). P99 latency < 400ms. Includes memory query, graph traversal, and filtering. Latency increases with graph depth and result count.</p>

<p><strong>Data Durability:</strong> 99.999999999% (11 nines) durability for stored impressions and patterns. Achieved through multi-region replication, automated backups (hourly snapshots retained for 7 days, daily snapshots retained for 30 days), and point-in-time recovery (RPO < 1 hour, RTO < 4 hours).</p>

<h3 id="observability">Observability</h3>

<p>Fabric Mind provides comprehensive observability for operational transparency and debugging.</p>

<p><strong>Trace ID Propagation:</strong> All API requests include a <code>trace_id</code> header that propagates through the entire request lifecycle (ingestion → impression → context assembly → NBA → execution). Trace IDs enable end-to-end request tracing and correlation across distributed components. Customers can inject their own trace IDs for integration with existing observability platforms (Datadog, New Relic, Honeycomb).</p>

<p><strong>Correlation and Audit Evidence:</strong> Every memory operation and action execution is logged with full provenance: timestamp, user/service identity, tenant ID, input parameters, output results, and trace ID. Audit logs are immutable and tamper-evident (cryptographic hashing with blockchain-style chaining). Logs are retained for 2 years by default (configurable up to 7 years for compliance).</p>

<p><strong>Metrics and Dashboards:</strong> Real-time metrics exposed via Prometheus-compatible endpoints and pre-built Grafana dashboards. Key metrics include: API request rate and latency (P50, P95, P99), memory store size and growth rate, pattern detection throughput, NBA recommendation acceptance rate, and governance policy violation count. Custom metrics can be exported to customer-managed observability platforms via webhook or pull-based integration.</p>
```

```
<h3 id="governance-enterprise">Governance</h3>

<p>Enterprise governance capabilities ensure compliance,
auditability, and operational control.</p>

<p><strong>Retention Policies:</strong> Configurable per tenant
with default 2-year retention for reinforced memories and 90-day
retention for unreinforced impressions. Retention enforcement
runs daily, automatically deleting expired memories. Supports
legal hold (freeze retention for litigation or investigation) and
data deletion requests (GDPR right to erasure).</p>

<p><strong>Decay Mechanisms:</strong> Memory confidence decays
exponentially over time unless reinforced by repeated exposure or
successful outcomes. Default decay half-life: 30 days
(configurable per tenant). Decay prevents stale memories from
influencing reasoning inappropriately.</p>

<p><strong>Freeze and Rollback:</strong> Emergency controls for
incident response and forensic analysis. Freeze blocks all memory
writes while preserving read access, enabling forensic
investigation without contamination. Rollback restores memory
state to a previous checkpoint (hourly snapshots available for 7
days). Both operations are logged for audit and require admin
privileges.</p>

<p><strong>Escalation Workflows:</strong> Configurable approval
workflows for high-risk actions. Supports multi-level escalation
(tier 1 → tier 2 → manager) with timeout policies (auto-deny
after 4 hours, auto-escalate after 1 hour). Escalation rules are
tenant-specific and version-controlled.</p>

<h3 id="billing-model">Billing Model</h3>

<p>Fabric Mind uses a transparent usage-based billing model with
predictable subscription tiers and metered add-ons.</p>

<p><strong>Subscription Tiers:</strong> Base subscription
includes platform access, standard SLOs, and usage quotas (10K
events/month, 5K context assemblies/month, 2K NBA
recommendations/month). Tiers scale with usage: Starter
($500/month), Professional ($2,500/month), Enterprise (custom
```

```
pricing). Annual commitments receive 15% discount.</p>

<p><strong>Usage Meters:</strong> Metered billing for usage
beyond subscription quotas. Pricing: $0.01 per event ingested,
$0.05 per context assembly, $0.10 per NBA recommendation, $0.02
per memory write (impression or pattern update). Usage is
aggregated monthly with overage charges billed in arrears.</p>

<p><strong>Enterprise Add-Ons:</strong> Private deployment
(Dedicated Tenant): +$5,000/month. Compliance pack (HIPAA,
FedRAMP, SOC 2 Type II): +$2,000/month. Extended retention (7
years): +$1,000/month. Premium support (24/7, 1-hour response
SLA): +$3,000/month. Custom integrations and professional
services: quoted separately.</p>


<h3 id="control-data-plane">Control Plane vs Data Plane</h3>

<p>Fabric Mind architecture separates control plane
(configuration, governance, admin operations) from data plane
(event processing, memory queries, action execution) to ensure
operational stability, security isolation, and independent
scaling.</p>

<div class="diagram-container">
  <svg width="760" height="400" viewBox="0 0 760 400"
xmlns="http://www.w3.org/2000/svg">
    <rect x="10" y="10" width="740" height="380" fill="#FFFFFF"
stroke="#D1D5DB" stroke-width="2" rx="8"/>
    <text x="380" y="35" font-size="14" fill="#111827" text-
anchor="middle" font-weight="700">Control Plane vs Data Plane
Architecture</text>

    <!-- Control Plane -->
    <rect x="40" y="60" width="320" height="300" fill="#FEF2F2"
stroke="#DC2626" stroke-width="2" rx="6"/>
    <text x="200" y="85" font-size="12" fill="#DC2626" text-
anchor="middle" font-weight="700">Control Plane</text>
    <text x="200" y="102" font-size="8" fill="#991B1B" text-
anchor="middle" font-style="italic">(Configuration & Governance)
</text>
```

```
    <!-- Control Plane Components -->
    <rect x="60" y="120" width="280" height="40" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="200" y="143" font-size="9" fill="#991B1B" text-
anchor="middle">Tenant Management & Provisioning</text>

    <rect x="60" y="170" width="280" height="40" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="200" y="193" font-size="9" fill="#991B1B" text-
anchor="middle">Governance Policy Configuration</text>

    <rect x="60" y="220" width="280" height="40" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="200" y="243" font-size="9" fill="#991B1B" text-
anchor="middle">Retention & Decay Rules</text>

    <rect x="60" y="270" width="280" height="40" fill="#FFF1F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="200" y="293" font-size="9" fill="#991B1B" text-
anchor="middle">Emergency Controls (Freeze, Rollback)</text>

    <rect x="60" y="320" width="280" height="30" fill="#FEF2F2"
stroke="#DC2626" stroke-width="1" rx="3"/>
    <text x="200" y="338" font-size="8" fill="#991B1B" text-
anchor="middle" font-weight="600">Isolated from data plane for
security</text>

    <!-- Data Plane -->
    <rect x="400" y="60" width="320" height="300" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="2" rx="6"/>
    <text x="560" y="85" font-size="12" fill="#1E40AF" text-
anchor="middle" font-weight="700">Data Plane</text>
    <text x="560" y="102" font-size="8" fill="#1E40AF" text-
anchor="middle" font-style="italic">(Event Processing &
Execution)</text>

    <!-- Data Plane Components -->
    <rect x="420" y="120" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="560" y="143" font-size="9" fill="#1E40AF" text-
anchor="middle">Event Ingestion & Impression Engine</text>
```

```
    <rect x="420" y="170" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="560" y="193" font-size="9" fill="#1E40AF" text-
anchor="middle">Memory Store & Context Assembly</text>

    <rect x="420" y="220" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="560" y="243" font-size="9" fill="#1E40AF" text-
anchor="middle">Pattern Detection & NBA Engine</text>

    <rect x="420" y="270" width="280" height="40" fill="#DBEAFE"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="560" y="293" font-size="9" fill="#1E40AF" text-
anchor="middle">Action Execution & Audit Logging</text>

    <rect x="420" y="320" width="280" height="30" fill="#EFF6FF"
stroke="#3B82F6" stroke-width="1" rx="3"/>
    <text x="560" y="338" font-size="8" fill="#1E40AF" text-
anchor="middle" font-weight="600">Scales independently for high
throughput</text>

    <!-- Separation Line -->
    <line x1="380" y1="60" x2="380" y2="360" stroke="#6B7280"
stroke-width="2" stroke-dasharray="5,5"/>

    <!-- Benefits -->
    <rect x="40" y="370" width="680" height="20" fill="#F9FAFB"
stroke="#D1D5DB" stroke-width="1" rx="4"/>
    <text x="380" y="384" font-size="8" fill="#374151" text-
anchor="middle" font-weight="600">Benefits: Security isolation |
Independent scaling | Operational stability | Compliance
separation</text>
  </svg>
</div>

<h2 id="final-recommendation">Final Recommendation: Adopting
Fabric Mind in Production</h2>

<h3 id="adoption-principle">Adoption Principle</h3>

<p>Fabric Mind should be adopted as foundational platform
infrastructure, not as a feature, agent framework, or prompt
```

```
system. Memory persistence, governance controls, and outcome
feedback loops are shared capabilities that must operate
consistently across all agentic workflows within the enterprise.
Treating memory as application-level state or embedding it within
individual agent implementations creates fragmentation, prevents
cross-session learning, and undermines the operational benefits
of persistent organizational memory.</p>

<p>Platform-level adoption ensures that all agents—regardless of
their domain, framework, or execution environment—share the same
memory substrate, governance policies, and outcome calibration
mechanisms. This architectural decision is not optional for
production-grade agentic AI. Without it, agents remain stateless,
learning resets with each session, and operational failures recur
without correction.</p>

<h3 id="adoption-sequence">Recommended Adoption Sequence</h3>

<p>Fabric Mind adoption follows a three-phase model designed to
minimize risk, validate behavior, and build confidence before
enabling autonomous execution. Each phase has clear entry
criteria, success metrics, and exit conditions.</p>

<p><strong>Phase 1 — Memory Instrumentation (Read-Only)</strong>
</p>

<p>In this phase, Fabric Mind ingests events, generates
impressions, stores memory, and assembles context, but does not
influence agent behavior. All recommendations remain in advisory
or shadow mode. The objective is to validate that memory
infrastructure operates reliably without changing production
workflows.</p>

<p>Activities:</p>
<ul>
  <li>Configure event ingestion from source systems (incident
alerts, support tickets, telemetry, user actions).</li>
  <li>Enable impression generation and semantic memory storage.
</li>
  <li>Integrate context assembly into agent workflows, passing
assembled context as additional input without modifying agent
logic.</li>
```

```
  <li>Run NBA engine in shadow mode, logging recommendations
without executing them.</li>
  <li>Monitor ingestion throughput, memory query latency, context
assembly accuracy, and shadow recommendation agreement rate.</li>
</ul>

<p>Success criteria:</p>
<ul>
  <li>Event ingestion operates at target throughput with &amp;lt;1%
error rate.</li>
  <li>Memory queries return results in &amp;lt;200ms at p95.</li>
  <li>Context assembly retrieves relevant impressions with
&amp;gt;80% relevance score (validated by human review).</li>
  <li>No degradation in agent performance or user experience.
</li>
  <li>Shadow recommendations logged for all eligible situations.
</li>
</ul>

<p>Exit condition: Infrastructure stability confirmed over 30
days of production traffic.</p>

<p><strong>Phase 2 — Pattern Learning & Recommendation</strong>
</p>

<p>In this phase, pattern detection activates, NBA
recommendations are surfaced to operators or agents in advisory
mode, and outcome feedback begins. The objective is to validate
that learned patterns reflect reality and that recommendations
improve decision quality.</p>

<p>Activities:</p>
<ul>
  <li>Enable pattern detection across accumulated memory.</li>
  <li>Surface NBA recommendations in agent UIs or workflows as
suggestions (not automated actions).</li>
  <li>Capture outcome labels (success, failure, neutral) for all
actions taken, whether recommended by NBA or chosen by operators.
</li>
  <li>Monitor pattern confidence, recommendation agreement rate,
and outcome correlation.</li>
  <li>Tune assembly policies (freshness window, confidence
```

```
threshold, graph depth) based on recommendation quality.</li>
</ul>

<p>Success criteria:</p>
<ul>
  <li>Pattern detection identifies recurring situations with
&gt;70% confidence and &gt;10 supporting observations.</li>
  <li>NBA recommendations achieve &gt;85% agreement rate with
operator decisions (measured over 1000+ actions).</li>
  <li>Repeated failure rate decreases by &gt;30% for situations
where NBA recommendations are followed.</li>
  <li>Operators report that recommendations provide useful
context and reduce diagnostic time.</li>
</ul>

<p>Exit condition: Recommendation quality validated over 60 days
with consistent agreement and outcome improvement.</p>

<p><strong>Phase 3 — Gated Automation</strong></p>

<p>In this phase, high-confidence NBA recommendations are
executed automatically within governance constraints. Automation
is limited to low-risk actions initially, with approval workflows
and rollback mechanisms enforced. The objective is to achieve
measurable operational improvement while preserving safety and
auditability.</p>

<p>Activities:</p>
<ul>
  <li>Define action risk classifications (low, medium, high) and
approval thresholds.</li>
  <li>Enable gated execution for low-risk actions with confidence
&gt;0.9 and pattern support &gt;20 observations.</li>
  <li>Enforce governance controls: approval workflows for
medium/high-risk actions, audit logging for all executions,
rollback on negative outcomes.</li>
  <li>Monitor execution rate, outcome distribution, escalation
frequency, and operational metrics (MTTR, AHT, cycle time).</li>
  <li>Expand automation scope incrementally based on outcome
validation.</li>
</ul>
```

```
<p>Success criteria:</p>
<ul>
  <li>Automated actions execute with &gt;95% success rate
(measured by outcome labels).</li>
  <li>Escalation rate remains &lt;5% of automated actions.</li>
  <li>Operational metrics improve: MTTR reduced by &gt;40%, AHT
reduced by &gt;25%, cycle time reduced by &gt;30%.</li>
  <li>No safety incidents or compliance violations attributable
to automated actions.</li>
  <li>Audit trails complete and accessible for all executions.
</li>
</ul>

<p>Exit condition: Automation operates reliably in production
with measurable business impact and maintained safety posture.
</p>

<h3 id="ownership-model">Ownership Model</h3>

<p>Fabric Mind requires clear ownership boundaries to operate
effectively at enterprise scale. The following model separates
platform responsibilities from application responsibilities while
ensuring accountability and operational clarity.</p>

<p><strong>Platform / Infrastructure Team:</strong> Owns Fabric
Mind deployment, configuration, and operational health.
Responsible for event ingestion reliability, memory store
performance, API availability, governance policy enforcement, and
compliance controls. Manages tenant provisioning, retention
policies, and platform upgrades. Provides observability
dashboards, SLA monitoring, and incident response for platform-
level issues.</p>

<p><strong>Application Teams:</strong> Own agent workflows, event
schema definitions, outcome labeling, and integration logic.
Responsible for configuring assembly policies, defining action
catalogs, setting risk classifications, and tuning recommendation
thresholds. Validate that context assembly retrieves relevant
memory for their use cases. Capture outcome feedback and report
recommendation quality issues to platform team.</p>

<p><strong>Security & Compliance:</strong> Own governance
```

```
policies, retention rules, data classification, and audit
requirements. Define approval workflows, escalation rules, and
emergency controls (freeze, rollback, circuit breakers). Review
audit logs, validate compliance with regulatory requirements
(GDPR, SOC2, HIPAA), and approve risk classifications for
automated actions.</p>

<p>This separation is necessary because memory infrastructure
must operate consistently across all applications, governance
must be enforced uniformly, and application teams must retain
autonomy over their agent behavior and outcome definitions.
Without clear ownership, memory becomes fragmented, governance
becomes inconsistent, and operational accountability erodes.</p>

<h3 id="definition-success">Definition of Success</h3>

<p>Fabric Mind succeeds in production when the following
conditions are observable and sustained:</p>

<ul>
  <li><strong>No repeated failures across sessions.</strong>
Agents do not retry the same failed action without recognizing
prior attempts. Diagnostic steps are not repeated when similar
situations recur. Resolved issues do not reappear without memory
of past resolutions.</li>
  <li><strong>Outcome-informed actions.</strong> Recommendations
reflect learned patterns from past outcomes, not just semantic
similarity. Actions that previously succeeded are prioritized.
Actions that previously failed are deprioritized or flagged for
review.</li>
  <li><strong>Confidence decay when conditions change.</strong>
Pattern confidence decreases when outcomes no longer match
predictions. Stale memories lose influence over time unless
reinforced. The system adapts to changing operational conditions
without manual retraining.</li>
  <li><strong>Auditable and reversible automation.</strong> Every
automated action is logged with provenance (context, pattern,
confidence, approval status). Rollback mechanisms execute
successfully when outcomes are negative. Audit trails are
complete and accessible for compliance review.</li>
  <li><strong>Learning compounds over time.</strong> Operational
metrics improve continuously as memory accumulates. Pattern
```

```
confidence increases with reinforcement. Context assembly becomes
more accurate as the memory graph grows. The system answers "Have
we seen this before?" with increasing reliability.</li>
</ul>

<p>These indicators are operational, not aspirational. They can
be measured through observability dashboards, outcome logs, and
operational metrics. If these conditions are not met, the
platform is not succeeding, and adoption should pause until root
causes are addressed.</p>

<h3 id="closing-statement">Closing Statement</h3>

<p>Fabric Mind complements foundation models, agent frameworks,
and context engineering—it does not replace them. Models provide
reasoning capability. Frameworks provide execution orchestration.
Context engineering provides inference-time information
retrieval. Fabric Mind provides what none of these address:
persistent, governed organizational memory with outcome feedback
loops.</p>

<p>Without memory, agentic AI remains stateless. Without
governance, automation remains unsafe. Without outcome feedback,
learning does not occur. Persistent, governed memory is not an
enhancement to production-grade agentic AI—it is the missing
prerequisite.</p>
```