# **1. Product Vision**

Build a modular, policy‑governed, multi‑agent reasoning platform that:

* Orchestrates heterogeneous LLMs and custom reasoning engines.
* Supports autonomous and collaborative agents (planning, debate, consensus, delegation).
* Enforces safety, policy, resource, and cost constraints in real time.
* Provides memory (short, episodic, long‑term), tool use, adaptive strategies, and persistent auditability.
* Scales to enterprise multi-tenancy with observability, governance, and extensibility via plugins.

Primary Value:

1. Faster experimentation with multi-agent strategies.
2. Safe deployment (policy + runtime monitors).
3. Data-driven iteration (evaluation harness + telemetry).

# **2. Core Use Cases**

|  |  |  |
| --- | --- | --- |
| **Category** | **Use Case** | **Description** |
| Single Agent | Task reasoning | An agent executes a prompt with adaptive style & policies. |
| Multi-Agent | Debate / brainstorming / consensus | Configurable round orchestration with convergence detection. |
| Workflow Automation | Task graph execution | Planner agent decomposes task into DAG of subtasks assigned to specialists. |
| Tool Augmentation | Retrieval & actions | Agents call tools (search, DB query, code executor, API) under policy. |
| Memory-Enriched Dialog | Knowledge retention | Agent retrieves episodic + semantic memory to ground responses. |
| Governance | Policy compliance | Keyword/regex/custom semantic filters + resource budgets + redaction. |
| Observability | Audit & replay | Persisted reasoning trees + violation logs + cost accounting. |
| Evaluation | Regression scoring | Benchmark tasks to compare model/agent versions. |

# **3. Functional Requirements (Concrete)**

* FR1: Create / update / run agents via REST & SDK.
* FR2: Multi-agent session creation: specify participants, mode (debate/brainstorm/consensus/workflow), max rounds, convergence policy.
* FR3: Agents must support sync + async inference and provider fallback (OpenAI, Anthropic, local).
* FR4: Policy monitors: per-turn + final-output evaluation; configurable via uploaded JSON/YAML.
* FR5: Tool interface: register tool with schema, rate limit, allow agent invocation via tool selection policy.
* FR6: Memory system:
  + Short-term: last N turns (bounded).
  + Episodic: persisted transcripts segment indexed by metadata.
  + Semantic: vector store (FAISS / Qdrant / Milvus / PGVector).
* FR7: Task Graph Execution:
  + Planner generates DAG with nodes {task\_id, prompt, dependencies, assigned\_agent}.
  + Scheduler executes nodes when dependencies complete, aggregates outputs.
* FR8: Session persistence: JSON + optional Postgres schema + object storage (for large artifacts).
* FR9: Observability: metrics (latency, tokens, violations), tracing (OpenTelemetry), structured logs.
* FR10: Cost tracking per provider + per tenant.
* FR11: Versioning: agent configs & policy bundles immutable with version tags.
* FR12: Evaluation harness trigger via API: run scenario set, store metrics & outcome diffs.
* FR13: RBAC: roles (admin, operator, auditor, tenant\_user).
* FR14: Redaction pipeline for export (strip sensitive fields).
* FR15: CLI + Python SDK + Web dashboard.

# **4. Non-Functional Requirements**

|  |  |
| --- | --- |
| **Attribute** | **Target** |
| Latency (single inference) | P50 < 1.2s (remote LLM dependent) |
| Horizontal Scalability | Stateless inference pods behind queue |
| Availability | 99.5% initial target |
| Audit Retention | 90 days hot, archive after |
| Security | JWT + OIDC integration; encrypted secrets |
| Data Privacy | Tenant isolation at DB schema or row-level |
| Throughput | > 100 concurrent sessions / node (depending on model) |
| Extensibility | New monitor or tool plugin in < 30 min dev |
| Fault Handling | Graceful fallback chain (primary → backup model → synthetic stub) |

# **5. High-Level Architecture (Textual Diagram)**

Code

+---------------- Web / SDK / CLI ----------------+

| REST / GraphQL

v

+------------------ API Gateway / Auth Layer ------------------+

| Rate limiting | JWT/OIDC | Request shaping | Version routing |

+----------------------+----------------------+-----------------+

|

v

+------------- Orchestrator Service ---------------+

| Session Manager | Dialogue Engine | DAG Planner |

| Convergence Check | Turn Scheduler | Tool Router |

+-------------------+-------------------------------+

| | |

| | |

v v v

+-----------+ +-------------+ +-------------+

| Agent Runtime (per agent) | | Tool Runner |

| Style, Policy, Memory fetch | | Exec sandbox|

+-----------+------------------+ +-------------+

|

v

+------------------ Inference Layer ------------------+

| Provider Adapters (OpenAI, Anthropic, Local LLM) |

| Retry / Fallback / Streaming |

+------------------+----------------------------------+

|

v

+------------- Observability & Governance --------------+

| Policy Monitor Bus | Violation Store | Metrics (TSDB)|

| Tracing (OTel) | Logs | Cost Engine | Alerting |

+----------+-----------------+--------------------------+

|

v

+------------------- Persistence Layer --------------------+

| Postgres (agents, sessions, DAGs, violations, costs) |

| Vector Store (semantic memory) |

| Object Storage (artifacts, raw transcripts) |

+---------------------------------------------------------+

# **6. Component Specifications**

|  |  |  |
| --- | --- | --- |
| **Component** | **Responsibilities** | **Tech** |
| API Gateway | AuthN/Z, rate limit, version routing | FastAPI or Envoy + FastAPI backend |
| Orchestrator | Session lifecycle, multi-agent control, scheduling | Python (FastAPI service) |
| Agent Runtime | Wraps DuetMindAgent + extension hooks | Python module |
| Provider Adapters | Normalize LLM calls (prompt, streaming, cost calculation) | Adapter classes |
| Policy Engine | Extend current rules: semantic classifier, embedding distance, numeric thresholds | Python + ONNX optional |
| Monitor Manager | Real-time evaluation pipeline (pre, mid, post) | Sync/async chain |
| Memory Service | Manages multi-tier memory (short, episodic, semantic) | Postgres + Vector DB |
| Tool Service | Validation, sandboxed exec (code / HTTP), caching | Python + Firecracker / subprocess isolation |
| DAG Planner | Generates & validates directed acyclic task graphs | Graph library (networkx) |
| Cost Engine | Token accounting, provider pricing table, anomaly detection | Background worker |
| Evaluation Harness | Batch run tasks, store metrics & regressions | Worker queue |
| Telemetry Stack | Metrics, traces, logs | Prometheus, Jaeger/OTel, Loki |
| Dashboard | Web UI for sessions, agents, policies, metrics | Next.js or React |
| Event Bus (optional) | Decoupled events (TURN\_COMPLETED, VIOLATION\_RAISED) | Kafka / NATS / Redis streams |

# **7. Data Model (Initial Postgres Schema)**

SQL

CREATE TABLE agent ( id UUID PRIMARY KEY, name TEXT UNIQUE NOT NULL, version TEXT NOT NULL, style JSONB NOT NULL, config JSONB, monitors JSONB, created\_at TIMESTAMPTZ DEFAULT now(), archived BOOLEAN DEFAULT FALSE ); CREATE TABLE session ( id UUID PRIMARY KEY, topic\_start TEXT, topic\_final TEXT, mode TEXT, strategy TEXT, converged BOOLEAN, rounds\_executed INT, participants TEXT[], -- agent names or IDs metadata JSONB, started\_at TIMESTAMPTZ DEFAULT now(), ended\_at TIMESTAMPTZ ); CREATE TABLE turn ( id UUID PRIMARY KEY, session\_id UUID REFERENCES session(id), round INT, agent\_name TEXT, prompt TEXT, content TEXT, confidence DOUBLE PRECISION, raw JSONB, created\_at TIMESTAMPTZ DEFAULT now() ); CREATE INDEX turn\_session\_round\_idx ON turn(session\_id, round); CREATE TABLE violation ( id UUID PRIMARY KEY, session\_id UUID REFERENCES session(id), turn\_id UUID REFERENCES turn(id), monitor\_name TEXT, severity TEXT, violation\_type TEXT, rationale TEXT, meta JSONB, created\_at TIMESTAMPTZ DEFAULT now() ); CREATE TABLE task\_graph ( id UUID PRIMARY KEY, session\_id UUID REFERENCES session(id), graph\_json JSONB, status TEXT, created\_at TIMESTAMPTZ DEFAULT now() ); CREATE TABLE cost\_event ( id UUID PRIMARY KEY, session\_id UUID, agent\_name TEXT, provider TEXT, model TEXT, tokens\_input INT, tokens\_output INT, cost\_usd NUMERIC(12,6), created\_at TIMESTAMPTZ DEFAULT now() ); CREATE TABLE memory\_episode ( id UUID PRIMARY KEY, agent\_name TEXT, embedding VECTOR(1536), -- if using pgvector content TEXT, metadata JSONB, created\_at TIMESTAMPTZ DEFAULT now() );

# **8. Memory Architecture**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Purpose** | **Implementation** |
| Short-term | Last K turns (window) | In-memory ring buffer (per session) |
| Episodic | Persisted conversation chunks | memory\_episode rows |
| Semantic | Vector similarity retrieval | Embeddings stored in pgvector / Qdrant |
| Structured Facts (future) | Entity-relation graph | Neo4j or typed Postgres tables |

Retrieval Flow:

1. Pre-inference hook → gather context: short-term tail + top N semantic matches (cosine sim threshold) + episodic matches by tags (e.g., “design-session”).
2. Summarize if token budget exceeded (rolling summarizer agent).

# **9. Message / Turn Schema (JSON Contract)**

JSON

{ "task\_id": "uuid", "agent": "Athena", "input": { "prompt": "Analyze X", "context": { "short\_term": ["..."], "semantic\_refs": [{"id": "e1", "score": 0.83}], "episodic\_refs": [] } }, "output": { "content": "Reasoned answer...", "confidence": 0.87, "style\_insights": ["Applying rigorous logical validation"], "tokens": {"input": 324, "output": 512} }, "monitors": [ { "name": "keyword\_guard", "status": "pass", "elapsed\_ms": 2.1 } ], "cost": { "provider": "openai", "model": "gpt-4o", "usd": 0.01234 }, "timing": { "started": 1736981000.123, "completed": 1736981001.001, "latency\_ms": 878 }, "version": "agent:Athena@1.0.0"}

# **10. API Endpoints (Representative)**

|  |  |  |
| --- | --- | --- |
| **Method** | **Path** | **Purpose** |
| POST | /agents | Create agent version |
| GET | /agents/{name} | Fetch agent config |
| POST | /sessions | Start multi-agent session |
| POST | /sessions/{id}/advance | Force round advance (manual control) |
| GET | /sessions/{id} | Session state & transcript summary |
| GET | /sessions/{id}/turns | Paginated turns |
| POST | /tasks/plan | Generate task DAG |
| POST | /tasks/execute | Execute DAG (returns execution id) |
| POST | /policies | Upload policy bundle |
| GET | /violations | Query violations (filters) |
| POST | /eval/run | Trigger evaluation suite |
| GET | /cost/summary | Cost aggregates |
| POST | /tools | Register tool |
| POST | /tools/invoke | Direct test invocation |
| GET | /metrics/health | Liveness & readiness |

Example: Create session body:

JSON

{ "topic": "Designing resilient edge network", "mode": "brainstorm", "strategy": "round\_robin", "agents": ["Athena@1.0.0", "Apollo@1.0.0"], "max\_rounds": 6, "convergence": true, "metadata": {"project": "edgeX"}}

# **11. Multi-Agent Orchestration Logic (Concrete Algorithm)**

Pseudo-flow for each round:

1. Load session config.
2. For each agent (strategy ordering):
   * Prepare prompt via template:
     + Mode directive + role hint + last R synthesized summary + differential context (avoid repetition).
   * Fetch memory context.
   * Invoke agent.generate\_reasoning\_tree() (sync/async).
   * Run monitors (pre-output & post-output).
   * Persist turn + cost + violations.
3. Update convergence state:
   * Maintain sliding window of last W contents.
   * Compute Jaccard of token sets; if ≥ threshold → mark converged.
4. If DAG mode:
   * Resolve ready nodes (dependencies satisfied).
   * Assign specialized agents & push to execution queue.
5. If converged or max rounds reached → finalize summary (final synthesis agent).
6. Persist session closure.

# **12. Task Graph Execution Spec**

DAG JSON Example:

JSON

{ "nodes": [ {"id": "root", "prompt": "Plan high availability design", "agent": "Planner", "deps": []}, {"id": "analysis", "prompt": "Analyze failure modes", "agent": "Athena", "deps": ["root"]}, {"id": "mitigations", "prompt": "Propose mitigations", "agent": "Apollo", "deps": ["analysis"]}, {"id": "summary", "prompt": "Integrate plan & mitigations", "agent": "Synthesizer", "deps": ["mitigations"]} ]}

Scheduler:

* Maintain ready\_set = { n | all deps complete }.
* For each ready node → push job to execution queue (e.g., Redis).
* Worker runs agent inference + monitors, updates node status (pending → running → done | failed).
* Partial failure strategy: retry K times, escalate to fallback agent, mark session degraded.

# **13. LLM Provider Adapter Contract**

Interface:

Python

class ProviderAdapter: def infer(self, model: str, prompt: str, \*\*opts) -> ProviderResult: ... def stream(self, model: str, prompt: str, \*\*opts) -> Iterable[Chunk]: ... def cost(self, tokens\_in: int, tokens\_out: int, model: str) -> float: ...

Supported adapters in Phase 1:

* OpenAI (chat + json mode)
* Anthropic (Claude)
* Local (vLLM / llamafile) Phase 2:
* Azure OpenAI
* HuggingFace Inference Endpoint
* Custom internal model

Fallback Chain Example: gpt-4o → if rate limited → gpt-4o-mini → if provider unreachable → local model → last resort synthetic placeholder with low confidence.

# **14. Tool System Specification**

Tool Registration:

JSON

{ "name": "web\_search", "input\_schema": {"type": "object", "properties": {"query": {"type": "string"}}, "required": ["query"]}, "rate\_limit\_per\_min": 30, "exec\_mode": "external\_api", "auth": {"type": "api\_key\_ref", "key\_id": "serpapi\_key"}, "allowed\_agents": ["Athena", "\*"]}

Invocation Flow:

1. Agent decides via internal heuristics or explicit tool selection policy.
2. Orchestrator validates (rate limit, authorization).
3. Tool Runner executes (sandbox if code).
4. Output cached (hash of input) if idempotent.
5. Output appended to agent context for next reasoning call.

# **15. Policy & Governance Extensions**

Add rule types:

* semantic\_similarity: block if cosine similarity above threshold to blacklisted embeddings.
* llm\_classifier: call a lightweight moderation model (distilled).
* numerical\_threshold: e.g., token count > limit.

Policy Bundle Example:

YAML

rules: - name: no\_pii type: regex severity: severe pattern: "(\\b\\d{3}-\\d{2}-\\d{4}\\b)" - name: toxicity\_screen type: llm\_classifier severity: major params: { model: "moderation-small", threshold: 0.78 } - name: semantic\_block\_list type: semantic\_similarity severity: severe params: { embedding\_set: "restricted\_corpora", max\_similarity: 0.92 }

# **16. Security & Isolation**

|  |  |
| --- | --- |
| **Aspect** | **Mechanism** |
| Auth | OIDC (Keycloak / Auth0) → JWT with tenant claim |
| Rate Limiting | Redis token bucket per tenant & per agent |
| Secret Management | Vault or AWS Secrets Manager; adapter fetch on demand |
| Tool Sandbox | Firecracker microVM or gVisor for untrusted code |
| Prompt Injection Mitigation | Pre-filter input + structural guard (whitelist macro expansion) |
| Data Isolation | Row-level security (RLS) in Postgres by tenant\_id |
| Export Controls | Redaction pipeline removes PII & secrets markers |
| Supply Chain | Dependency scanning (Snyk, Trivy) in CI |

# **17. Deployment Topology**

Environments: dev → staging → prod

Kubernetes Services:

* api-gateway (FastAPI)
* orchestrator-service
* inference-workers (HPA on queue depth)
* tool-runner
* evaluation-runner
* vector-store (managed or in-cluster)
* postgres (managed)
* redis (queue + cache)
* kafka/nats (optional event backbone)
* prometheus + grafana + tempo/jaeger + loki

Ingress: NGINX / Envoy  
CDN (optional) for static dashboard.

# **18. Observability (Concrete Metrics)**

|  |  |  |
| --- | --- | --- |
| **Metric** | **Labels** | **Source** |
| inference\_latency\_ms | model, provider, agent | Adapter |
| tokens\_consumed\_total | model, provider, tenant | Adapter |
| violations\_total | severity, rule\_name | Monitor pipeline |
| cost\_usd\_total | provider, model, tenant | Cost engine |
| session\_rounds | mode, converged | Orchestrator |
| tool\_invocations\_total | tool\_name, status | Tool runner |
| dag\_node\_duration\_ms | agent, node\_type | DAG scheduler |

Tracing Spans:

* session.round.turn
* agent.inference
* monitor.sequence
* tool.invoke

Log Structure: JSON lines with correlation\_id = session\_id.

# **19. Evaluation Harness**

Config Example:

YAML

suite: "baseline\_design\_eval"scenarios: - name: "resilience\_planning" mode: "brainstorm" agents: ["Athena", "Apollo"] topic: "Edge network partition recovery" success\_criteria: - type: keyword\_presence must\_include: ["redundancy", "failover"] - type: semantic\_score reference\_id: "ref\_doc\_12" min\_score: 0.78 - name: "ethical\_debate" mode: "debate" agents: ["Athena", "Hermes"] topic: "Autonomous swarm decision hierarchy"

Output stored with per-scenario metrics:

* coverage\_score
* novelty\_score (embedding distance from reference corpus)
* coherence\_score (LLM evaluation model)
* rule\_violations\_count

# **20. Cost Tracking**

Flow:

1. Adapter returns token counts (or estimates).
2. Pricing table (JSON) loaded at startup; formula applied.
3. Persist cost\_event.
4. Aggregator job rolls up hourly/daily totals per tenant & model.
5. Alerts if spend velocity > configured threshold.

# **21. CI/CD Pipeline**

Steps:

1. Lint + type check (ruff + mypy).
2. Unit tests (pytest) + coverage gate > 85%.
3. Security scan (bandit / trivy).
4. Contract tests for provider adapters (mocked).
5. Integration tests: spin ephemeral Postgres + vector store.
6. Evaluation regression: run mini suite → block if key scenario degrades > threshold.
7. Build Docker images (tag = git sha + semantic version).
8. Deploy via ArgoCD or Flux (staging → manual approve → prod).
9. Post-deploy smoke test: health + test inference.

# **22. Risk Register (Initial)**

|  |  |  |
| --- | --- | --- |
| **Risk** | **Impact** | **Mitigation** |
| Provider outage | Degraded service | Fallback chain + circuit breaker |
| Policy false positives | User frustration | Tiered severity + allow override token (audited) |
| Token cost spike | Budget overrun | Spend alerts + automatic model downgrade |
| Memory bloat | Latency degrade | Compression (summaries) & TTL pruning |
| Tool injection | Data exfiltration | Strict schema validation + sandbox |
| Convergence stall | Long sessions | Hard caps + stagnation detection |
| Prompt leakage of secrets | Compliance breach | Secret scanning in outputs + redaction |

# **23. Phase Roadmap (Quarter-Level)**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Duration** | **Goals** |
| P0 (Foundations) | Weeks 1-4 | Core agent runtime refactor, provider adapters (OpenAI/local), sessions, monitors v1 (keyword/regex/resource), persistence basics |
| P1 (Multi-Agent & Policies) | Weeks 5-10 | Debate/brainstorm modes, convergence engine, policy bundle loader, dashboard MVP, cost tracking v1 |
| P2 (Tools & Memory) | Weeks 11-16 | Tool system, vector semantic memory, episodic store, retrieval integration |
| P3 (Task Graph & Eval) | Weeks 17-22 | DAG planner/scheduler, evaluation harness, semantic policy rules |
| P4 (Governance & Scaling) | Weeks 23-28 | Advanced monitors (semantic/classifier), RBAC, multi-tenancy isolation, autoscaling tuning |
| P5 (Optimization & UX) | Weeks 29-34 | UI polish, real-time streaming, plugin SDK, fallback orchestration refinement |
| P6 (Enterprise Hardening) | Weeks 35-40 | Auditing dashboards, spend anomaly detection, export controls, SOC2-aligned logging |

# **24. First 6-Week Sprint Breakdown (Detailed)**

### **Week 1**

* Extract current DuetMindAgent into core/agent.py.
* Implement provider adapter interface + OpenAI adapter.
* Set up FastAPI skeleton + auth stub (API key temp).
* Define DB schema migrations (Alembic).

### **Week 2**

* Implement single-agent inference endpoint /agents/{name}/infer.
* Add monitor pipeline abstraction (refactor existing MonitorFactory into plugin folder).
* Add cost event recording (hardcoded pricing table).

### **Week 3**

* Multi-agent session endpoints /sessions.
* Round orchestration service with convergence detection (reuse existing logic).
* Persist turns & violations.
* Basic metrics (Prometheus) + structured logging.

### **Week 4**

* Resource monitor & policy file loader endpoint /policies.
* Introduce dashboard skeleton (list sessions, violations).
* Add fallback provider config & circuit breaker pattern.

### **Week 5**

* Convergence visualization in dashboard.
* Add tooling for embedding store (choose pgvector) and create ingestion endpoint.
* Memory retrieval prototype (semantic + short-term merge) behind feature flag.

### **Week 6**

* Harden error handling (CognitiveFault serialization).
* Add evaluation harness scaffold (single scenario run).
* Internal load test + performance tuning (connection pooling, async path).
* Prepare P1 planning (retrospective adjustments).

# **25. Future Enhancements (Beyond P6)**

* Streaming tool invocation with partial reasoning adaptation.
* Agent reputation / reinforcement scoring loop.
* Federated memory sync across clusters.
* On-device partial execution (edge inference).
* Structured reasoning graphs (expose as graph API).
* Adaptive strategy selection (choose debate vs consensus based on topic classification).

# **26. File / Module Layout (Proposed)**

Code

/src

/api

agents.py

sessions.py

policies.py

tools.py

eval.py

/core

agent.py

monitors/

base.py

keyword.py

regex.py

resource.py

semantic.py

policies/

loader.py

memory/

store.py

retriever.py

provider/

base.py

openai\_adapter.py

local\_adapter.py

orchestration/

session\_manager.py

round\_engine.py

dag\_planner.py

scheduler.py

tools/

registry.py

executor.py

cost/

pricing.py

recorder.py

eval/

runner.py

metrics.py

/infra

db.py

migrations/

/observability

metrics.py

tracing.py

/cli

main.py

/tests

/unit

/integration

/perf

# **27. Concrete Extensions to Current DuetMindAgent**

|  |  |
| --- | --- |
| **Current** | **Extension Needed** |
| Monitors (keyword/regex/resource/custom) | Add semantic + classifier monitors |
| Synchronous & async reasoning | Add streaming variant with incremental monitors |
| Basic dialogue | Introduce pluggable strategy (class-based Strategy interface) |
| Style influence | Externalize style transformers in config registry |
| Persistence (JSON files) | Replace with DB persistence abstraction |
| Policy loaded from file | Add remote versioned policy store + hot reload |
| Convergence (lexical overlap) | Add semantic embedding convergence + stagnation detection |

# **28. Testing Strategy**

|  |  |
| --- | --- |
| **Layer** | **Tests** |
| Unit | Monitors, policy parsing, memory retrieval |
| Contract | Provider adapters (mock responses) |
| Integration | Multi-agent session lifecycle |
| Load | 500 concurrent sessions with stub model |
| Chaos | Provider latency injection, tool failure |
| Security | Injection attempts in prompts, tool sandbox breakouts |
| Regression | Evaluation suite diff vs baseline corpus |

# **29. Immediate Action Checklist (You Can Start Now)**

1. Spin a new repo structure as above.
2. Isolate DuetMindAgent core logic → convert monitors to interface-based.
3. Implement ProviderAdapter base + OpenAI adapter.
4. Build FastAPI skeleton with /health & /agents basic endpoints.
5. Write Alembic migrations from schema above.
6. Create first evaluation scenario YAML and harness stub.
7. Implement metrics exporter (Prometheus).
8. Add basic cost accumulator.

# **30. Next Step Offer**

If you tell me:

* Preferred vector DB (pgvector vs Qdrant).
* Primary provider (OpenAI vs local).
* Target deployment platform (AWS/GCP/on-prem).

I can generate:

* Concrete docker-compose for dev.
* Initial code skeleton with file stubs.
* Sample policy and agent config pack.