SPEC-001 — Epistemology Engine (Canonical Technical Specification)

SPEC-001 — Epistemology Engine (Canonical Technical Specification) 1.0 Background The Epistemology Engine is a grounded knowledge-processing platform designed to ingest heterogeneous artifacts (documents, images, web pages), normalize them against a curated ontology, and produce verifiable, audit-ready outputs. It targets zero-hallucination generation through strict grounding, validation, and a closed feedback loop with Human-in-the-Loop (HIL) review. The system uses Postgres + pgvector as the authoritative source of truth, with Notion (or similar) mirrored as a convenience surface—never as the system of record. Orchestration is handled by a resilient control plane (Prefect/Dagster) to ensure idempotency, retries, and lineage. Model/version drift is managed via a model registry and dual-index read paths to preserve vector neighborhoods during migration. Cost is controlled via ROI-driven vision tiling, adaptive DPI, token budgets, and feature caching. A reviewer console and lightweight mobile micro-UI ensure field adoption and structured feedback (Step 10B), closing the loop for continuous adaptation.

Primary Deployment & Data Residency: AWS (us-east-1), containerized services on EKS, RDS for Postgres 17, pgbouncer, S3 for blobs.

Initial Corpus Modalities & Languages: PDF, PNG/JPG scans, and HTML; languages EN first, ES as secondary (with language detection on ingest).

2.0 Requirements (MoSCoW)

Must Have

- Single Source of Truth (DB-first): All authoritative writes land in Postgres; Notion is a mirror only. [Ref: §3.2, §3.4, Step 10A]
- Provenance & Auditability: Every mutation recorded in ops.write_sets with run_id, reversible via ops.rollback_run. [Ref: §3.2 ops.*, §3.4]
- Resilient Orchestration: Prefect/Dagster control plane with idempotent tasks, retries, checkpoints, lineage. [Ref: §3.3]
- Grounded Generation: All model outputs must carry citations to core.sources/pages (auto-fail if missing). [Ref: Steps 6–8]
- Validation Gates: Schema checks, rule-based checks, and LLM-as-judge rubric before publish; failures enqueue HIL tasks. [Ref: §3.2 eval.*, Steps 8–9]
- Reviewer Console + Mobile Micro-UI: Approve/reject with span-level comments; events in hil.feedback_events. [Ref: §3.4, Steps 10B–10C]
- Cost Controls: Token/DPI budgets enforced by policy; vision/embedding caches keyed by (content_hash, model_key). [Ref: §3.2 registry.policies, Steps 2,4]
- Vector Retrieval at Scale: pgvector with ANN index (IVFFlat/HNSW where available), connection pooling (pgbouncer). [Ref: §3.2 core.embeddings]
- Ontology Versioning: ontology.topic_taxonomy versioned; reads pinned to version; dual-index path during migrations. [Ref: §3.2 ontology.*, §2.6]
- Multilingual Ingest (EN→ES): Language detection at ingest; pipeline traces language for routing/eval. [Ref: Steps 1–2]
- Security Baseline: AWS IAM least-privilege; KMS-encrypted S3; RDS encryption; TLS in transit; audit logs. [Ref: §5 infra, CI/CD policies]
- SLOs v1: Q&A; RAG p95 ≤ 20s end-to-end; ingestion throughput ≥ 500 pages/hour/worker at 200–300 DPI adaptive. [Ref: Steps 1–7]

Should Have

- Hybrid Search: Keyword + vector fusion with recency/topic filters. [Ref: Step 6]
- Shadow Reads During Migrations: Dual-read to old/new indexes with consistency metrics. [Ref: §2.6, Step 10C]
- FinOps Telemetry: finops.usage_events with cost/tokens per run/step/model; monthly caps per scope. [Ref: §3.2 finops.*]
- Policy Registry: Centralized DPI/routing/LLM policy rollout with canaries and rollback. [Ref: §3.2 registry.policies]
- Docs/Runbooks: MkDocs site (operator playbooks, incident runbooks, DAG catalog). [Ref: §5 docs/]
- Observability: Traces/metrics/logs with task/run correlation IDs; error budgets & alerts. [Ref: §3.3, §5 infra/]
- Access Control: Reviewer roles/queues; audit trails for HIL actions. [Ref: §3.4 hil.*]
- Redaction Utilities: Optional PII masking before mirroring to Notion. [Ref: Step 10A]

Could Have

- Active Learning Loops: Use HIL feedback to auto-curate eval sets and adjust thresholds. [Ref: Steps 10B–10C]

- Autoscaling Workers: Queue-depth-based scaling for parse/embed tasks. [Ref: §3.3]
- Panel of Judges: Multi-model validation voting for high-risk outputs. [Ref: Step 8]
- Query Planner: Cost-aware retrieval planner (token/time budgets) for large prompts. [Ref: Step 6]

Won’t Have (v1)

- End-user Notion authoring as write-source. (Mirror is read-only.) [Ref: §2.5]
- Realtime streaming ingestion/answers. Batch/near-real-time only for v1.
- Fine-tuning foundation models. Use prompt/policy routing and evaluation first.

3.0 Method

1) Canonical Architecture (runtime view)

```
@startuml
skinparam packageStyle rectangle
actor "Operator" as Op
actor "Reviewer" as Rev
actor "API Client" as Api
package "Control Plane (Orchestration)" {
[Orchestrator] as ORCH
}
package "Data Plane" {
[API Gateway / FastAPI] as API
[Ingest Worker]
[OCR/Parse Worker]
[Chunker]
[Embedder]
[Indexer]
[RAG Service]
[Validator]
[Publisher]
[Mirror Sync]
[Adaptation Jobs]
}
package "Storage & Infra" {
database "RDS PostgreSQL 17\n+ pgvector ≥0.8" as PG
[S3 Artifacts\n(raw PDFs, images, HTML, tiles)]
[Pgbouncer]
}
package "HIL Surface" {
[Reviewer Console (Web)]
[Mobile Micro-UI]
}
Op --> ORCH : deploy, runbooks
Api --> API : queries, admin
ORCH --> Ingest Worker
Ingest Worker --> S3 Artifacts
Ingest Worker --> PG : core.sources, documents
ORCH --> OCR/Parse Worker
OCR/Parse Worker --> S3 Artifacts
OCR/Parse Worker --> PG : core.pages
ORCH --> Chunker --> PG : core.chunks
ORCH --> Embedder --> PG : core.embeddings (HNSW)
ORCH --> Indexer --> PG
API --> RAG Service --> PG : vector + hybrid search
RAG Service --> Validator --> PG : eval.scores
Validator --> Publisher --> PG : publish mutations
Publisher --> Mirror Sync : Notion mirror
Rev --> Reviewer Console --> PG : hil.feedback_events
Reviewer Console --> Mobile Micro-UI
Adaptation Jobs --> PG : registry.policies, ontology.version
@enduml
```

Orchestrator: Dagster 1.11.x

Database: PostgreSQL 17 on RDS + pgvector ≥0.8

Indexing: HNSW indexes for high-recall ANN.

2) Data Model (DDL)

```
-- === Extensions (migration V0001) ===
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "vector"; -- pgvector
-- === Schemas ===
CREATE SCHEMA IF NOT EXISTS core;
CREATE SCHEMA IF NOT EXISTS ontology;
CREATE SCHEMA IF NOT EXISTS ops;
CREATE SCHEMA IF NOT EXISTS eval;
CREATE SCHEMA IF NOT EXISTS finops;
CREATE SCHEMA IF NOT EXISTS hil;
CREATE SCHEMA IF NOT EXISTS registry;
-- === Ontology & Registry ===
CREATE TABLE ontology.topic_taxonomy (
taxonomy_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
version TEXT UNIQUE NOT NULL,
body JSONB NOT NULL,
checksum BYTEA NOT NULL,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE registry.models (
model_key TEXT PRIMARY KEY, -- e.g., "text-embed-3-large@2025-05"
family TEXT, -- "openai", "bge", "clip"
modality TEXT, -- "text", "vision", "multimodal"
dims INT, -- embedding dimension if applicable
provider TEXT,
released_at TIMESTAMPTZ,
deprecated_at TIMESTAMPTZ
);
CREATE TABLE registry.policies (
policy_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
name TEXT UNIQUE NOT NULL, -- e.g., "vision_v1"
body JSONB NOT NULL, -- DPI rules, token caps, routing
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
-- === Sources & Documents ===
CREATE TABLE core.sources (
source_id UUID PRIMARY KEY,
uri TEXT NOT NULL,
mime_type TEXT,
title TEXT,
metadata JSONB,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE core.documents (
doc_id UUID PRIMARY KEY,
source_id UUID NOT NULL REFERENCES core.sources(source_id),
external_ref TEXT,
lang TEXT,
status TEXT NOT NULL DEFAULT 'ingested', -- ingested|parsed|chunked|indexed|published
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE core.pages (
page_id UUID PRIMARY KEY,
doc_id UUID NOT NULL REFERENCES core.documents(doc_id) ON DELETE CASCADE,
page_num INT NOT NULL,
dpi INT,
text TEXT, -- OCR/plaintext
vision_features JSONB, -- cached detections
content_hash BYTEA NOT NULL, -- sha256
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
UNIQUE (doc_id, page_num)
);
-- === Chunks & Embeddings ===
CREATE TABLE core.chunks (
chunk_id UUID PRIMARY KEY,
doc_id UUID NOT NULL REFERENCES core.documents(doc_id) ON DELETE CASCADE,
page_id UUID REFERENCES core.pages(page_id),
span JSONB, -- offsets/coords
text TEXT NOT NULL,
```

```sql
topic_hint TEXT,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE core.embeddings (
embedding_id UUID PRIMARY KEY,
chunk_id UUID NOT NULL REFERENCES core.chunks(chunk_id) ON DELETE CASCADE,
model_key TEXT NOT NULL REFERENCES registry.models(model_key),
dims INT NOT NULL,
vec VECTOR NOT NULL,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
-- HNSW index (tune m and ef_construction per scale)
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_embeddings_hnsw
ON core.embeddings USING hnsw (vec) WITH (m = 16, ef_construction = 200);
-- Optional hybrid text search helpers
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_chunks_trgm ON core.chunks USING gin (text gin_trgm_ops);
-- === Topic links ===
CREATE TABLE core.chunk_topics (
chunk_id UUID NOT NULL REFERENCES core.chunks(chunk_id) ON DELETE CASCADE,
taxonomy_version TEXT NOT NULL,
topic_path TEXT NOT NULL,
score REAL NOT NULL,
PRIMARY KEY (chunk_id, taxonomy_version, topic_path)
);
-- === Ops: runs, steps, write sets, mirror state ===
CREATE TABLE ops.runs (
run_id UUID PRIMARY KEY,
pipeline TEXT NOT NULL, -- e.g., "ingest-index-publish"
started_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
finished_at TIMESTAMPTZ,
status TEXT NOT NULL DEFAULT 'running', -- running|succeeded|failed|rolled_back
metadata JSONB
);
CREATE TABLE ops.run_steps (
run_id UUID NOT NULL REFERENCES ops.runs(run_id) ON DELETE CASCADE,
step_name TEXT NOT NULL,
started_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
finished_at TIMESTAMPTZ,
status TEXT NOT NULL DEFAULT 'running',
info JSONB,
PRIMARY KEY (run_id, step_name)
);
CREATE TABLE ops.write_sets (
write_set_id UUID PRIMARY KEY,
run_id UUID NOT NULL REFERENCES ops.runs(run_id),
table_name TEXT NOT NULL,
pk JSONB NOT NULL, -- {cols, vals}
before JSONB,
after JSONB,
op TEXT NOT NULL, -- INSERT|UPDATE|DELETE
applied_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE ops.sync_state (
mirror TEXT NOT NULL, -- "notion"
entity_table TEXT NOT NULL, -- "core.documents"
entity_pk JSONB NOT NULL,
mirror_id TEXT, -- Notion page_id
mirror_rev TEXT,
last_synced_at TIMESTAMPTZ,
PRIMARY KEY (mirror, entity_table, entity_pk)
);
-- === Eval & FinOps (MVP) ===
CREATE TABLE eval.test_suites (
suite_id UUID PRIMARY KEY,
name TEXT UNIQUE NOT NULL,
description TEXT,
assertions JSONB NOT NULL, -- inline specs for MVP
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE eval.scores (
score_id UUID PRIMARY KEY,
suite_id UUID REFERENCES eval.test_suites(suite_id),
run_id UUID REFERENCES ops.runs(run_id),
metrics JSONB NOT NULL,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE TABLE finops.usage_events (
event_id UUID PRIMARY KEY,
run_id UUID REFERENCES ops.runs(run_id),
model_key TEXT REFERENCES registry.models(model_key),
tokens_prompt BIGINT DEFAULT 0,
tokens_completion BIGINT DEFAULT 0,
cost_usd NUMERIC(12,6) DEFAULT 0,
meta JSONB,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
-- === HIL ===
CREATE TABLE hil.feedback_events (
feedback_id UUID PRIMARY KEY,
entity_table TEXT NOT NULL,
entity_pk JSONB NOT NULL,
reviewer_id TEXT,
channel TEXT, -- "console", "mobile"
label TEXT, -- "approve", "reject", "needs_attention"
payload JSONB,
created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

3) Algorithms & Policies (MVP, deterministic and cost-aware) 3.1 Ingest & Vision Policy (adaptive DPI + tiling)

DPI selection: If mime = pdf and page entropy (text density estimate) ≥ threshold → 200 DPI; else 300 DPI for diagrams/tables. Respect per-policy cap (e.g., 300 DPI max).

Tiling: For pages > A4 at ≥300 DPI, tile into 2×2 or 3×3, overlap 10–15% for OCR continuity.

Cache key: sha256(content_bytes) + ":" + policy_rev + ":" + model_key.

3.2 OCR & Parsing

OCR engine: Pluggable (e.g., AWS Textract). Must emit bbox spans and confidence per token/line.

Normalization: Store plain text in core.pages.text; keep vision_features (tables, headers) as JSONB.

3.3 Chunking

Rule-first, semantic-second: Split by headings/table boundaries from vision_features, then enforce size band: P50 ≈ 700–1,000 chars; P95 ≤ 2,000. Store spans ({start,end} or bbox refs).

3.4 Embedding & Indexing

Write (chunk_id, model_key, dims, vec) to core.embeddings.

Index: HNSW with m=16, ef_construction=200; online tune ef_search at query time.

3.5 Hybrid Retrieval (vector ⊕ keyword ⊕ filters)

Candidate set (K): top-K by vector (cosine/IP), WHERE taxonomy_version = pinned and optional topic/date filters.

Rerank: BM25/Trigram score on core.chunks.text.

Blended score: $S = \alpha * sim\_vec + \beta * bm25\_norm + \gamma * recency\_boost$. Start $\alpha=0.6$, $\beta=0.3$, $\gamma=0.1$.

```
 * SQL sketch (server-side fusion):
   WITH vec AS (
  SELECT c.chunk_id, 1 - (e.vec <=> :qvec) AS vscore      -- cosine sim -> similarity
  FROM core.embeddings e
  JOIN core.chunks c ON c.chunk_id = e.chunk_id
  WHERE e.model_key = :model_key
  ORDER BY e.vec <-> :qvec
  LIMIT 200
),
kw AS (
  SELECT chunk_id, ts_rank_cd(to_tsvector('english', text), plainto_tsquery(:q)) AS kscore
  FROM core.chunks
  WHERE text ILIKE '%' || :kw || '%'
  LIMIT 500
)
SELECT c.chunk_id, (0.6*vscore + 0.3*COALESCE(kscore,0) + 0.1*recency_boost(c.doc_id)) AS score
FROM vec v
JOIN core.chunks c ON c.chunk_id = v.chunk_id
LEFT JOIN kw ON kw.chunk_id = c.chunk_id
ORDER BY score DESC
LIMIT 20;
```

3.6 Grounded Generation (schema-constrained)

Prompt contract: Model receives only retrieved chunks (with citations), the task schema, and forbidden behaviors (no unverifiable claims).

Output format: YAML/JSON with fields: claims[] {text, citation{source_id,page_id,offset}}, confidence, limitations.

3.7 Validation Gates

Schema validator: strict JSON schema.

Citation validator: every claim must cite ≥1 chunk; page/offset must exist.

LLM-as-judge: rubric: factual consistency, coverage; thresholded.

Failure path: create hil.feedback_events; block publish.

3.8 Publish & Mirror

Publish: write approved artifact rows; tag core.documents.status='published'.

Mirror: ops.sync_state tracks Notion page_id + mirror_rev to avoid drift; mirror is non-authoritative.

3.9 Adaptation

Consume hil.feedback_events to update registry.policies or stage re-embedding via dual-index path.

4) Orchestration (DAGs, idempotency, retries)

```
@startuml
start
:ingest(source_uri) -> core.sources, core.documents;
:ocr_parse(doc_id, policy);
:chunk(doc_id, policy_rev);
:embed(doc_id, model_key);
:index_build(doc_id);
:retrieve(query, taxonomy_version);
:generate(schema_id);
:validate(suite_id);
if (valid?) then (yes)
:publish();
:mirror_notion();
else (no)
:enqueue_HIL();
endif
:adaptation_loop();
```

```
stop
@enduml
```

Idempotency keys:

ingest: (source_uri, sha256)

ocr_parse: (doc_id, policy_rev)

embed: (doc_id, model_key)

generate: (query_hash, schema_id, taxonomy_version)

Retries/backoff: S3 and Notion sync with exponential backoff; dead-letter queues.

Orchestrator: Dagster 1.11.x

5) Reviewer Console & Mobile Micro-UI (API contract)

Endpoints (FastAPI, JWT via OIDC/Okta):

POST /v1/hil/feedback → writes hil.feedback_events

GET /v1/review/queue?queue=spec_review → filtered tasks

POST /v1/review/claim → optimistic lock, assign task

POST /v1/review/resolve → approve|reject|needs_attention + notes

AuthN/Z: OIDC login; roles reviewer, lead_reviewer, admin. SSO required.

6) FinOps controls (enforced at runtime)

Policy-driven budgets: registry.policies['finops_v1'] = { monthly_caps: { pipeline: 500.00 }, on_overflow: 'stop_and_alert' }

Token accounting: each LLM/embedding call logs finops.usage_events with model_key and exact tokens/cost.

7) Rollback mechanics (surgical & auditable)

All mutations wrapped in a run_id.

To revert a bad run: SELECT ops.rollback_run(:run_id); (replays ops.write_sets in reverse).

8) Deployment blueprint (AWS us-east-1)

Infra: RDS PostgreSQL 17 (Multi-AZ) + pgbouncer; S3 with Object Lock (compliance mode).

Compute: EKS for workers and API (HPA on queue depth).

Security: AWS Secrets Manager + IRSA; KMS encryption; ACM certs on ALB; SSO (Okta).

Ops: Automated snapshots + PITR; CloudWatch + OpenTelemetry (traces tied to run_id).

9) Example policy (YAML)

```
   version: vision_v1
max_dpi: 300
text_density_hi: 0.65
tiling:
  enable: true
  grid_by_longest_side:
    - threshold_px: 2200
      grid: [2, 2]
    - threshold_px: 3300
      grid: [3, 3]
cache_key_fields: [content_hash, policy_rev, model_key]
roi:
  table_detector: on
  math_detector: off
```

10) Minimal service interfaces (Python)

```
# src/clients/vector_store.py
class VectorStore:
    def upsert(self, chunk_id: str, vec: list[float], model_key: str, dims: int) -> None: ...
    def knn(self, qvec: list[float], model_key: str, k: int, filters: dict) -> list[str]: ...

# src/pipelines/ingest_flow.py (Dagster task/ops)
@op
def ingest(uri: str) -> UUID: ...

@op
def ocr_parse(doc_id: UUID, policy_rev: str) -> None: ...

@op
def embed(doc_id: UUID, model_key: str) -> None: ...
```

4.0 Implementation

This section turns the approved architecture into build steps. 0) Baseline stack & conventions

- Runtime: Python 3.12
- Package management: uv or pip-tools; lock files committed.
- API framework: FastAPI (+ Uvicorn).
- Orchestrator: Dagster 1.11.x
- DB: AWS RDS PostgreSQL 17 with pgvector; pgbouncer in front.
- Artifacts: S3 with Object Lock (compliance mode), lifecycle 365 days.
- Auth: OIDC (Okta) → JWT → FastAPI dependencies.
- Embedding: OpenAI text-embedding-3 family (registry-configurable).
- Observability: OpenTelemetry traces; JSON logs; CloudWatch metrics/dashboards.

1) Repository scaffolding

- Create the 11-point repo structure (as previously defined).
- pyproject.toml (excerpt)

[project] name = "epistemology-engine" requires-python = ">=3.12" dependencies = ["fastapi>=0.115", "uvicorn[standard]>=0.30", "pydantic>=2.8", "sqlalchemy>=2.0", "psycopg[binary]>=3.2", "pgvector>=0.2", "dagster>=1.11", "dagster-webserver>=1.11", "httpx>=0.27", "openai>=1.43", "google-generativeai", # For Gemini "boto3>=1.34", "tenacity>=8.3", "opentelemetry-sdk>=1.26", "opentelemetry-instrumentation-fastapi>=0.48b0", "python-json-logger>=2.0", "python-dotenv>=1.0", "langdetect>=1.0.9"]

- .github/workflows/ci.yml (high level)
- Jobs: lint (ruff), typecheck (mypy), unit tests (pytest), db-migration check (spin ephemeral Postgres 17 + pgvector), build Docker images, push to ECR, deploy to EKS (on main with tag).

2) Database bring-up

- Provision RDS Postgres 17 (Multi-AZ); enable vector extension.
- Deploy pgbouncer.
- Apply migration V0001 from db/migrations/ (DDL from Method §2).
- Seed:
- ontology.topic_taxonomy v1 from /config/topic_taxonomy.json.
- registry.models with text-embedding-3-large (dims=3072) and ...-small (dims=1536).
- registry.policies with vision_v1, finops_v1.

3) Dagster project

- Files: src/engine/pipelines/repo.py, jobs.py, assets/ (ingest.py, ocr_parse.py, chunk.py, embed.py, index.py, retrieve.py, generate.py, validate.py, publish.py, mirror.py, adapt.py).
- Patterns: Every op accepts an idempotency key, emits a run_id, writes to ops.run_steps, and wraps DB mutations with write_sets.log().

4) API layer (RAG + HIL endpoints)

- FastAPI app at src/engine/web/app.py:
- GET /healthz
- POST /v1/query → body: {query, filters: {taxonomy_version, topics[], date_range}, top_k}

POST /v1/hil/feedback → writes hil.feedback_events.

GET /v1/review/queue / POST /v1/review/claim / POST /v1/review/resolve.

Auth: OIDC middleware validating JWT via Okta JWKS.

5) OCR/Parsing adapters

Dev/local: Tesseract, pdfminer.six.

Prod toggle: Pluggable OCR provider (e.g., AWS Textract) behind an OCRClient interface.

Output: text (plain), vision_features (JSONB), content_hash, dpi, page_num.

6) Retrieval & generation

Vector store: Postgres/pgvector; HNSW index.

Hybrid: SQL fusion as in Method §3.5; $\alpha/\beta/\gamma$ configurable in registry.policies.

Templating: Jinja2 for prompts; output validator (JSON Schema).

7) Validation gates

Schema gate: strict JSON schema.

Citation gate: each claim must map to existing core.pages offsets.

Judge gate: rubric-scored; threshold in eval.test_suites.

On fail: create hil.tasks / hil.feedback_events; block publish.

8) Publish & Notion mirror

Publish: status flip + materialized views.

Mirror: reconcile using ops.sync_state (DB → Notion); mirror is read-only.

PII guard: optional redaction pass before mirroring.

9) FinOps wrapper

All OpenAI/Gemini calls go through a client wrapper that logs to finops.usage_events and enforces caps from registry.policies.

10) Observability & SLOs

Tracing: OTel FastAPI & Dagster instrumentors (propagate run_id).

Dashboards:

p95 end-to-end latency (answers) ≤ 20s.

p95 retrieval latency ≤ 1.5s.

Ingest throughput ≥ 500 pages/hour/worker.

Alarms: breach → PagerDuty/Slack.

11) Security controls

Network: EKS private subnets; RDS in private; ALB public only for API.

Secrets: AWS Secrets Manager + IRSA.

KMS: Encrypt S3, RDS, and logs.

SSO: Okta app with roles; JWT verified per request.

12) CI/CD

Pipelines:

check: lint/type/tests; migration check.

build: Docker build (multi-stage) → ECR.

deploy: Helm to EKS (API + workers + Dagster).

eval-gate: run eval suites; require ≥ threshold to merge to main.

5.0 Milestones

M0 — Foundation & CI (Week 0–1)

Deliverables: Repo scaffold; Dagster boilerplate; FastAPI skeleton; Dockerfiles; CI pipelines.

Exit criteria: CI green on PRs; dev can run API + Dagster locally.

M1 — Database & Ingest (Week 1–2)

Deliverables: RDS ready; V0001 migration applied; seed taxonomy/models/policies; ingest + OCR/parse ops.

Exit criteria: Upload 50-page sample corpus; pages parsed with content_hash; S3 artifacts retained.

M2 — Chunk + Embed + Index (Week 2–3)

Deliverables: Chunker (rule-first), embedding op (OpenAI t-e-3), HNSW index; hybrid search query.

Exit criteria: KNN latency p95 $\leq$ 1.5s on sample; chunk size P50 700–1,000 chars.

M3 — RAG + Validation (Week 3–4)

Deliverables: /v1/query endpoint; prompt templates; schema/citation gates; judge gate v1.

Exit criteria: End-to-end answer p95 $\leq$ 20s on golden set; failed validations correctly enqueue HIL tasks.

M4 — Publish + Notion Mirror (Week 4–5)

Deliverables: Publish stage; Notion mirroring with ops.sync_state & mirror_rev; PII redaction toggle.

Exit criteria: DB authoritative $\rightarrow$ Notion reflects within 5 min; drift detector shows 0 unresolved diffs.

M5 — HIL Console + Mobile Micro-UI (Week 5–6)

Deliverables: Reviewer queue, claim/resolve, span comments; OIDC SSO; audit logs.

Exit criteria: Reviewer performs approve/reject from phone in < 30s median; feedback stored in hil.feedback_events.

M6 — FinOps + Policies + Observability (Week 6–7)

Deliverables: Token/cost logging; monthly caps; SLO dashboards/alerts; runbooks.

Exit criteria: Budget breach test halts high-cost ops and pages on-call; dashboards show all SLOs.

M7 — Hardening & GA (Week 7–8)

Deliverables: Load test, backup/restore drill, chaos test on worker restarts; DR docs; security review.

Exit criteria: PITR validated; failure of an AZ doesn’t drop SLOs beyond error budget; GA tag.

6.0 Gathering Results

A. Offline evaluation (pre-merge gate)

Golden sets: Curate labeled Q/A and extraction tasks per topic.

Metrics: factual consistency, citation coverage, completeness, precision@k/recall@k for retrieval.

Thresholds: defined in eval.test_suites; CI blocks merges that regress.

B. Online quality (production)

SLO monitoring: p95 answer ≤ 20s; retrieval ≤ 1.5s; ingestion ≥ 500 pph/worker.

Canary releases: roll policy/model changes to 10% traffic; compare judge/implicit metrics.

Drift monitors: embedding neighborhood shift; topic distribution deltas.

C. HIL loop effectiveness

Adoption KPIs: median review time; % auto-approved vs HIL-required; reviewer agreement rate.

Learning signals: use hil.feedback_events to auto-augment eval sets.

D. FinOps & ROI

Cost per deliverable: $/answer, $/100 pages ingested; budget variance per month.

Policy impact: demonstrate savings from caching, adaptive DPI, and token caps.

E. Governance & Integrity

Rollback drills: quarterly ops.rollback_run exercise on staging; report MTTD/MTTR.

Mirror integrity: weekly diff report (DB vs Notion) with zero unresolved items as target.

Reporting cadence:

Weekly: SLO/FinOps dashboard review; bug triage.

Bi-weekly: Model/policy review, HIL insights, taxonomy changes.

Monthly: Post-release quality report, cost analysis, and roadmap adjustments.