Semantic Evaluation System

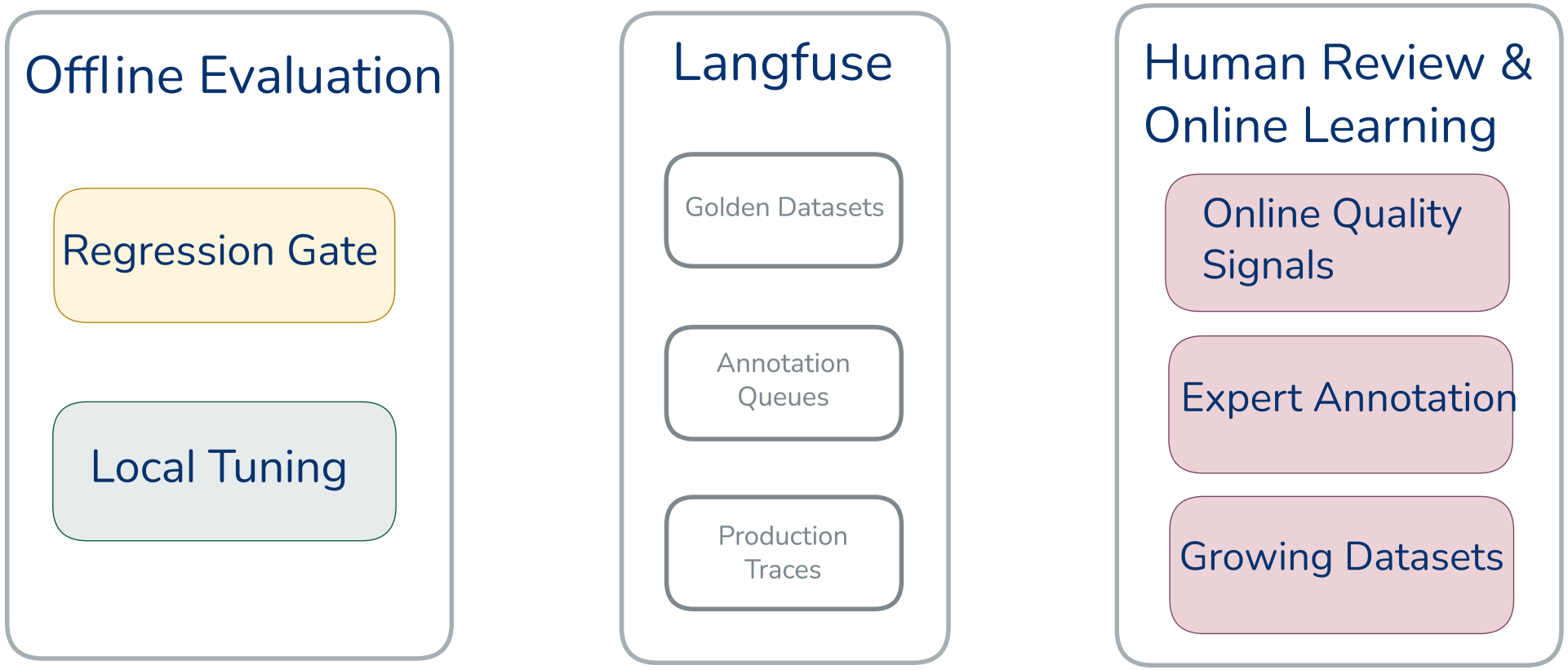
How to measure, gate, and continuously improve your AI applications

A shared system for observability, expert review, automated quality gating, and iterative improvement of LLM-powered applications

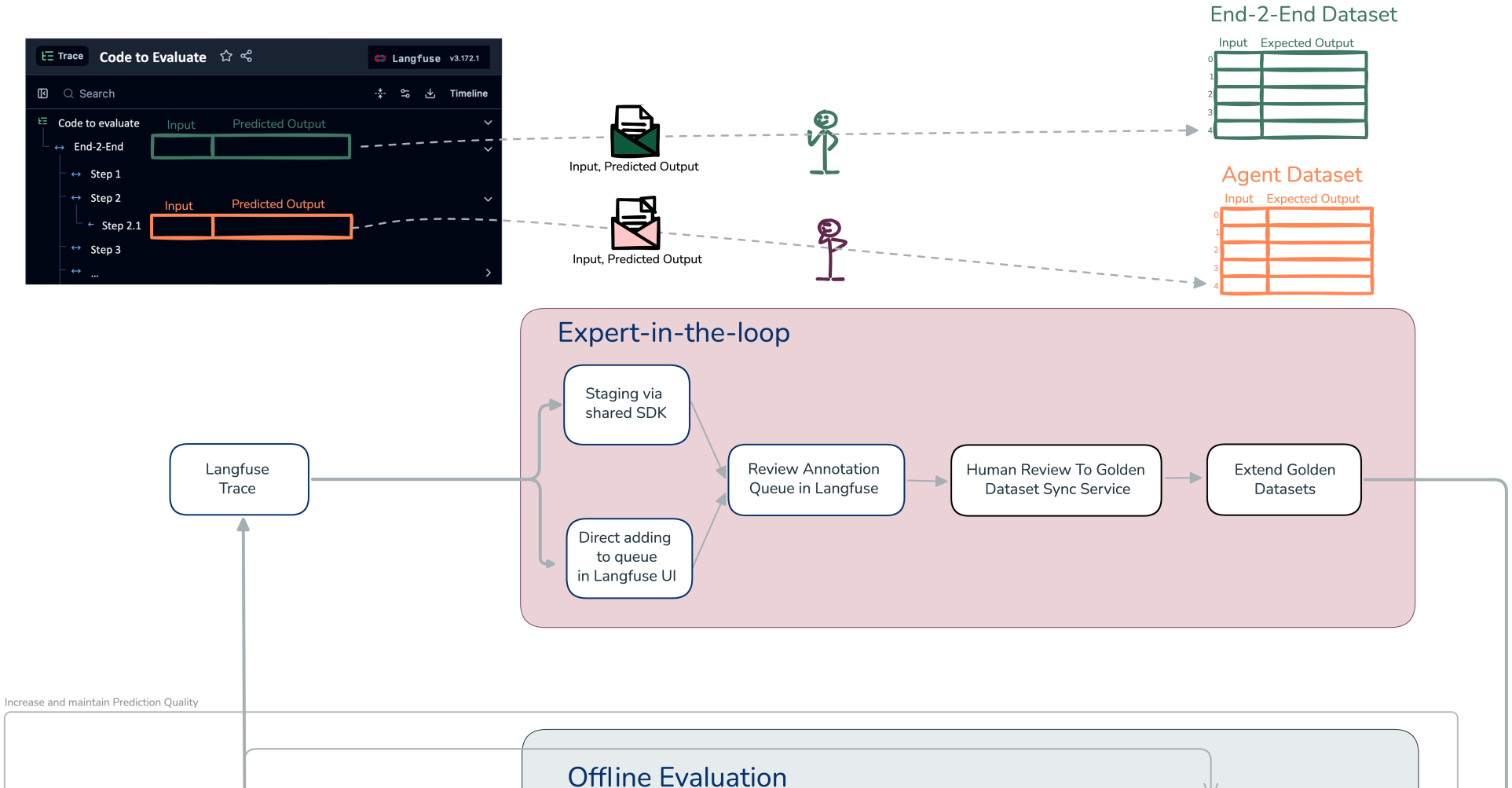
Why traditional quality assurance breaks down for AI applications

- LLM outputs are open-ended and non-deterministic. The same input can produce different outputs, and there is no single correct answer to assert against. Traditional unit tests do not apply.
- Manual quality evaluation — an engineer reviewing a sample of outputs, or sending edge cases to a domain expert by email — is expensive, slow, and produces nothing reusable. Expert judgment stays in an inbox.
- Every Ramboll team building an LLM workflow today rebuilds the same loose evaluation scripts from scratch, missing the shared tooling and shared learning that already exists.

Name for overview slide



End-To-End Evaluation Cycle



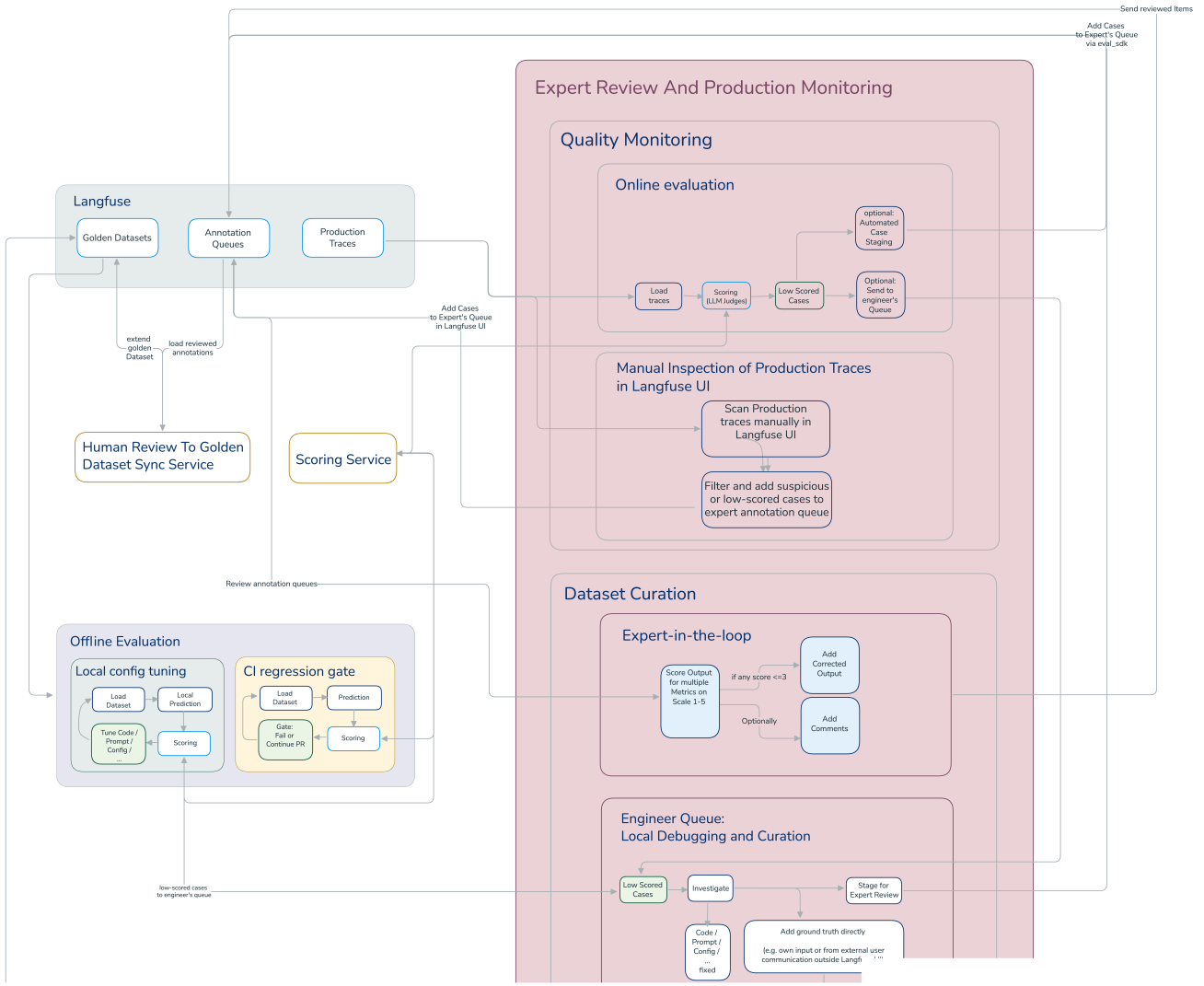
One Shared Backbone, Three Loops

Prediction remains pod-specific. Scoring and review-sync are shared services.

Langfuse is the system of record for traces, expert annotation queues, golden datasets, and published offline-evaluation runs.

Loop	Trigger	Purpose
Local tuning	Developer runs experiments	Optimize code, prompts, and configuration against golden datasets, write local artifacts, and publish comparison runs back to Langfuse
Regression gating	PR / CI Pipeline	Validate code or prompt changes through the same central scoring service and release-facing gate policy
Expert Review And Production Monitoring	Manually after trace inspection in Langfuse UI, or stage eval cases via the shared SDK. PLANNED Auto-route low-scored LLM-judge outputs to annotation queues for expert review.	Curate corrections, monitor production quality, and scale golden dataset growth

One Shared Backbone, Three Loops



Local tuning, CI gating, and expert review and production monitoring are not separate workflows.

They all connect through Langfuse as the shared evaluation backbone for traces,
golden datasets, annotation queues, and published offline-evaluation runs.

Golden datasets are human-curated ground truth. LLM judges evaluate predictions *against* that truth during

Responsibility Boundaries

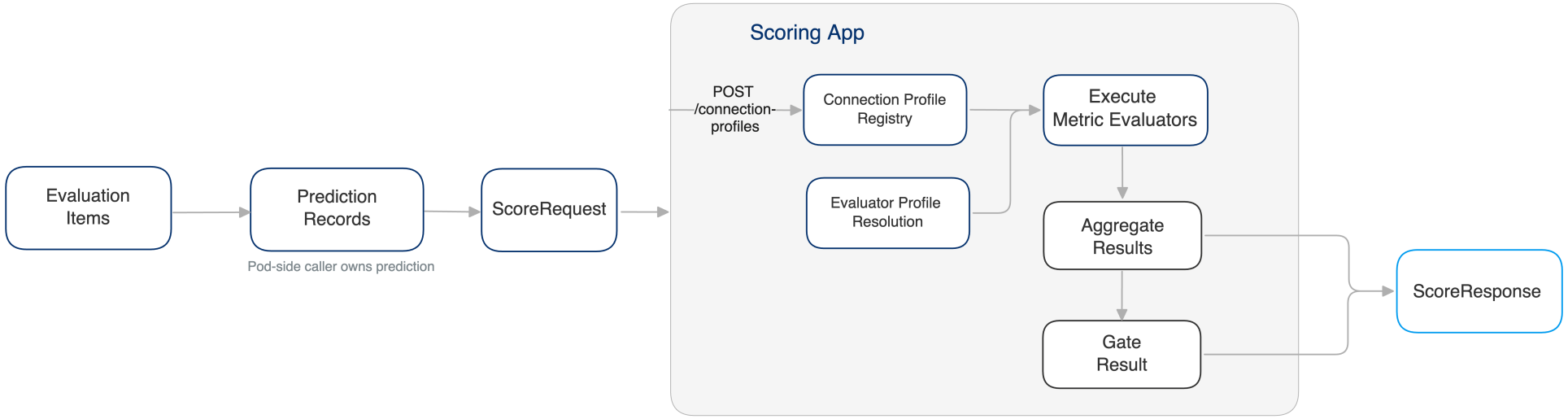
Concern	Pod / Engineer Surfaces	Shared Evaluation SDK	Shared Cloud Services	Langfuse
Prediction execution	Runs Code to Evaluate locally, in CI, or in pod cloud runtime	Prediction and evaluation contracts	-	Stores resulting traces
Trace-to-Review Flow	Chooses traces in Langfuse UI or stages eval cases via SDK	Staging and publication helpers	-	Stores seeded datasets and annotation queues
Scoring	Offline Evaluation sends scoring requests and consumes gate decisions	Scoring client, gate configuration, request and response schemas	`scoring_app` routes requests through the evaluator catalogue, resolves scoring connection profiles for LLM-judge backends, aggregates, and gates	Stores dataset runs and published scores
Human Review to Golden Dataset Sync	Registers queue bindings and triggers sync jobs	Binding and job contracts, client, normalization primitives	`review_sync_app` enforces the review policy, executes binding-driven merges, persists job results with checkpoints and cleanup audits, and publishes promoted records with reviewer comments and human review scores	Review UI, reviewed artifacts, queues, and golden datasets
Evaluator registration	Submits named rubric via `POST /v1/evaluators/requests`; inspects pending submissions via `GET /v1/evaluators/requests`	Registration contracts and submission helpers	`scoring_app` stores pending submissions; eval core team accepts rubrics to the catalogue; accepted entries load as `custom_rubric/{name}` after deployment	-

Offline Evaluation Reporting

Concern	Current behavior
Artifacts first	Shared runner writes the local JSON and JSONL artifact set before Langfuse publication starts
Publication surface	Shared runner publishes per-item scores, aggregate scores, and gate result back to Langfuse
Dataset-run identity	Langfuse dataset runs are dataset-scoped, so one execution may emit multiple dataset runs that share one execution name
Run type label	Each publication is tagged with a `run_type`: `ci_gate` (CI/PR trigger), `local_tuning` (developer experiment), or `ad_hoc` (any other invocation) — used to filter and compare runs by loop in Langfuse
Failure behavior	Publication failure fails the CLI even if the scoring gate passed
Caller boundary	Pod callers submit scoring inputs only; evaluator execution and provider selection stay inside `scoring_app`
Who publishes to Langfuse	The shared offline-evaluation runner in `eval_sdk` publishes dataset runs and scores. `scoring_app` returns results only — it does not contact Langfuse

Scoring Service

The service does not run the app / agent / code to evaluate. The evaluator catalogue routes each metric to one of three execution paths: built-in deterministic checks, external-SDK-backed evaluators (DeepEval, Azure AI Evaluation), or custom rubric evaluators — which accept either an inline rubric from the scoring request or a named rubric from the accepted registry. LLM-based paths resolve a service-owned scoring connection profile.



Scoring Service: Pod-Driven Evaluator Registration

A validated inline rubric can be promoted to a named catalogue entry available to all pods.

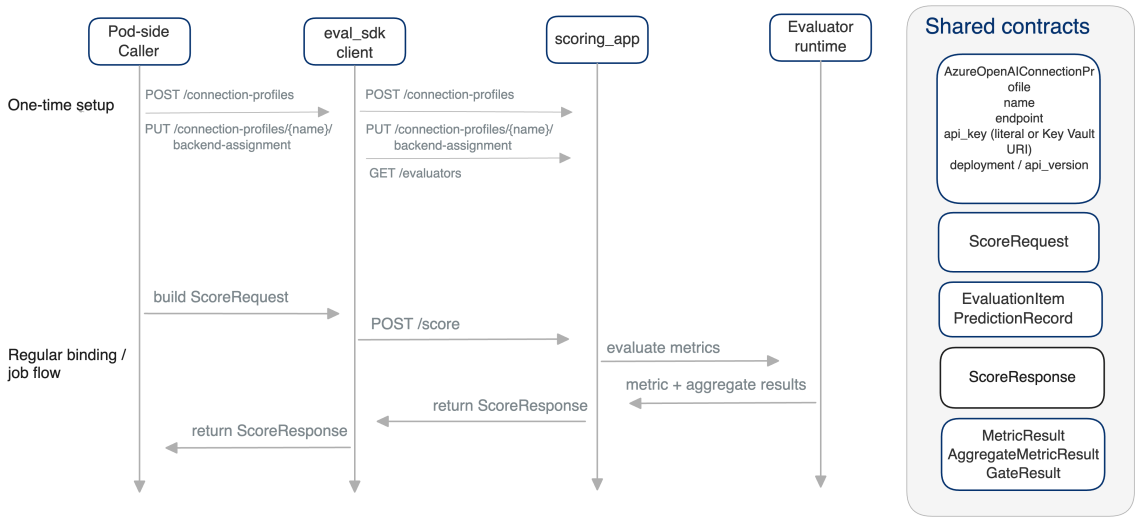
Stage	Who	Action	Outcome
Develop	Pod engineer	Iterate with `framework: custom_rubric`, `evaluator: inline`, rubric in `evaluator_params`	Pod-local metric; no server-side state; identical results in dev, CI, and cloud
Submit	Pod engineer	`POST /v1/evaluators/requests` with rubric payload and `X-API-Key`; re-submit to overwrite while pending	Submission written to pending queue on the scoring service
Review	Eval core team	`GET /v1/evaluators/requests` — inspect and triage pending submissions	Accept or reject decision
Accept	Eval core team	Commit rubric JSON to `scoring_service/custom_evaluators/accepted/`	Rubric committed to the scoring service repository
Deploy	Eval core team	Deploy the scoring service	Accepted rubric loaded as `custom_rubric/{name}` catalogue entry; available to all pods
Use	Any pod	`framework: custom_rubric`, `evaluator: {name}` in any manifest	Named evaluator shared across pods without copying the rubric definition

Scoring Service: LLM Connection Profiles

Concern	Current behavior
Config path	Profiles are registered at runtime via `POST /v1/connection-profiles`. The service persists them to `{OFFLINE_SCORING_STATE_DIR}/connection_profiles.json`. No static file path is needed — the HTTP API is the only registration surface.
Document shape	Each `AzureOpenAIConnectionProfile` carries `name`, `endpoint`, `api_key`, `deployment`, and `api_version`. `api_key` may be a literal string or an Azure Key Vault secret URI. Backend assignments (`PUT /v1/evaluator-backends/{backend}/connection-profile`) map each evaluator backend key to a registered profile name, and `GET /v1/evaluator-backends` reads the map back. No read route returns `api_key`.
Current backend keys	`deepeval_llm_judge`, `azure_ai_evaluation_llm_judge`, and `custom_rubric_llm_judge` — each resolves through the service-owned profile registry
Failure mode	Missing or invalid profile configuration fails before evaluator execution starts
Caller boundary	Callers keep sending scoring requests only. They do not send provider credentials inline for normal operation

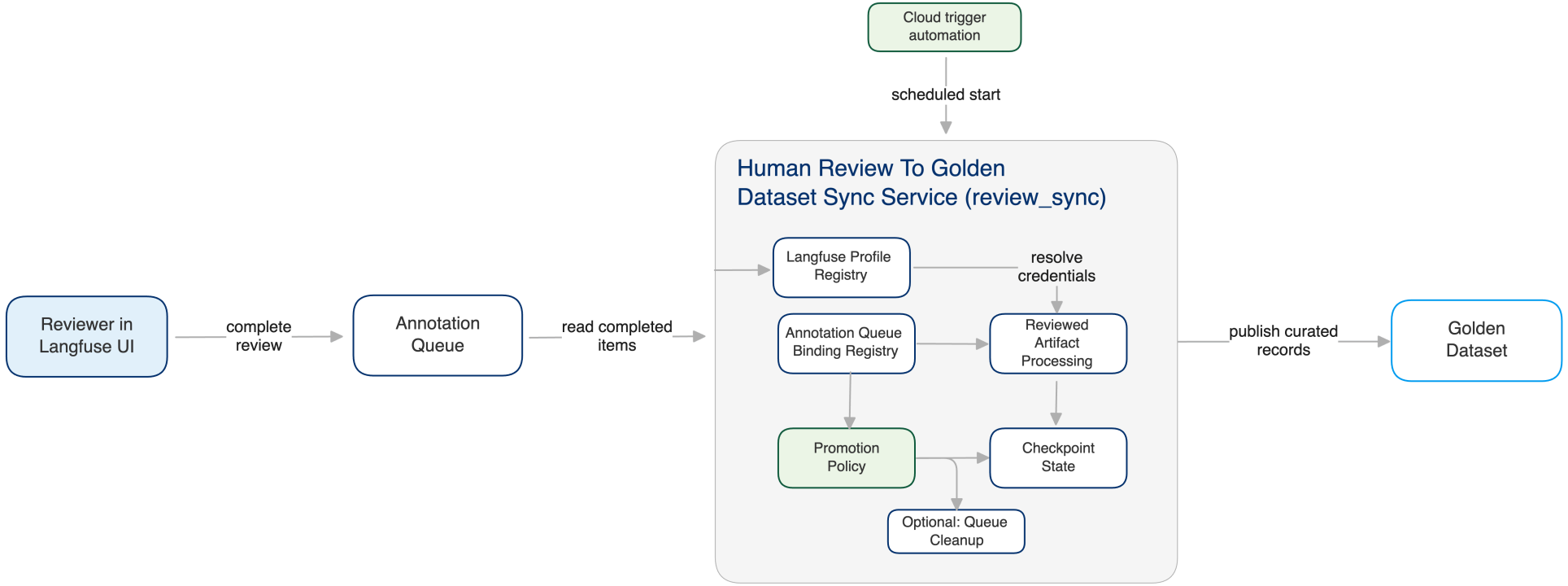
Scoring Service: Communication And Contracts

- Pod-side callers build `ScoreRequest` and call `POST /score` through eval_sdk's scoring client
- Request fields: `items`, `predictions`, `evaluators`, `gate`, optional `scopes`
- `scoring_app` resolves evaluator backends to scoring connection profiles internally. Callers never send provider credentials
- Response fields: `metric_results`, `aggregate_results`, `gate_result`, `item_count_by_scope`



Human Review To Golden Dataset Sync Service

The service enforces a configurable review policy, executes binding-driven merges, persists job results with checkpoint tracking, and publishes promoted records with reviewer comments and human review scores. Queue cleanup — removing completed queue items after promotion — is per-binding and disabled by default. The service manages its own Langfuse credentials via connection profiles registered separately from scoring connection profiles.

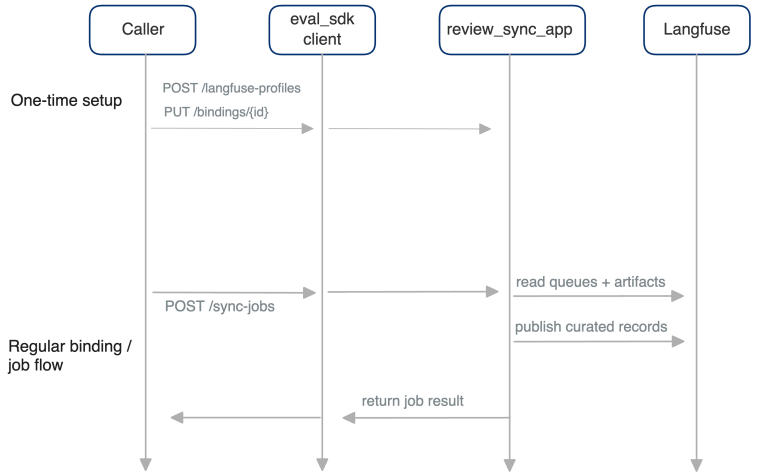


Human Review to Golden Dataset Sync Service: LangfuseConnection Profiles

Concern	Current behavior
Config path	Profiles are registered at runtime via `POST /v1/langfuse-profiles`. The service persists them to `{REVIEW_SYNC_STATE_DIR}/langfuse_profiles.json`. No static file path is needed — the HTTP API is the only registration surface.
Document shape	Each `LangfuseConnectionProfile` carries `profile_id`, `base_url`, `public_key`, and `secret_key`. `public_key` and `secret_key` may be literal strings or Azure Key Vault secret URIs.
Binding requirement	Every `QueueToDatasetBinding` must reference a registered `langfuse_profile_id`. The review sync service resolves the Langfuse credentials for that binding at sync time — callers never send credentials inline.
Failure mode	A binding that references an unknown `langfuse_profile_id` fails at sync-job execution time, not at binding registration time.
Caller boundary	Pods register profiles once and then reference `profile_id` in every binding. Review sync credentials are separate from scoring connection profiles and managed independently.

Human Review to Golden Dataset Sync Service: Communication And Contracts

- Manual pod-side or cloud automation
  register a `QueueToDatasetBinding`
  with `PUT /v1/bindings/{binding_id}`
  via shared client in evaluation SDK;
  each binding must reference a registered
  `langfuse_profile_id`
- Sync execution starts
  with `POST /v1/sync-jobs`
  carrying `JobRequest`
  (`binding_id` , optional `dry_run` ,
  `max_items` , `page_size`)
  via shared client in evaluation SDK
- The service persists sync job results, enforces the review policy, and publishes reviewed records plus reviewer comments to golden datasets
- Binding-scoped checkpoints, queue cleanup retries, and



Target Architecture And Deployment

