Versioned Prompt Provenance: A Substrate for Behavioral Tracking in Large Language Model Applications

mowa

TECHNICAL DESIGN NOTE · REGISTRY DESIGN

ABSTRACT

Prompt-controlled large language model applications typically lack a durable, queryable record of how a particular prompt version actually behaved in deployment. This note describes the versioned prompt registry implemented by mowa and the per-version metadata model that anchors it. Each prompt artifact detected in a source repository becomes a tracked entity with append-only version history, commit-level provenance, authorship, and a coarse health signal derived from an LLM auditor. We outline the data model that exists today, the operational properties it provides, and the further behavioral dimensions — refusal rate, output-length distribution, semantic consistency — that this substrate is designed to support as future work.

Motivation

Enterprise teams increasingly govern model behavior through prompts. Prompt text is edited as code, but the operational record that links a prompt revision to its downstream behavior is rarely first-class. Source repositories carry the artifact; production logs carry the behavior; nothing in between holds a stable, version-addressable reference that connects the two.

mowa is built around the observation that the prompt artifact must be treated as a versioned object with the same care as a database migration or an API contract. The registry described here is the persistence layer that makes this possible. It is deliberately scoped: it records prompt identity, lineage, authorship, and a small set of editorial-quality signals. It does not yet record sampled production behavior. Both halves are needed for a complete behavioral picture; this note is about the half that exists.

Data Model

Each tracked prompt has two associated tables. The **prompts** table holds the current addressable state — name, file path, container type (embedded fragment or dedicated file), source location, current health score, and pointers to two version rows: the version mowa considers current, and the version last observed on

GitHub. The **prompt_versions** table is append-only and stores the full content of every change, with content hash, sequential version number, parent lineage, commit SHA, commit message, author, and a source label that classifies the change (`initial_import`, `github_sync`, `suggestion`, `manual`, `rollback`).

Suggestion-sourced versions carry the pull request URL in the commit-SHA field as a deliberate overload, allowing the merge webhook to locate the row and promote it once the PR is merged. This is the mechanism by which mowa-side edits and GitHub-side history converge on a single timeline.

Provenance and Drift

Because the current-version pointer and the GitHub-version pointer are tracked independently, the system can recognize three states: *in sync* (the two pointers match), *drift* (the current version is ahead of what GitHub knows), and *behind* (an external commit landed and the local current pointer has not yet caught up). The editor surfaces these states as advisory banners — a pull action and a merge action are exposed, but the system does not block on either.

When the GitHub webhook reports a push to the default branch, the system re-resolves the prompt content against the new file using a layered strategy: whole-file match, position-anchored splice using surrounding code as before/after context, and an LLM extraction pass as a last resort. The chosen strategy is recorded in the version row and in the GitHub-content cache so that future reads carry their own justification.

Behavioral Signal Today

Each version carries a single integer **health** score in the closed interval [0, 100], computed by an LLM auditor against the prompt content and a stored issue summary. The score is coarse and intentionally so: it is calibrated for a reviewer skimming a pull request, not for automated gating. The auditor also produces a free-text **aiSummary** that explains what the score reflects, alternative phrasings the auditor would consider, and a suggested name for the prompt.

These two fields — score and summary — are the only behavioral signals stored per version today. They are derived from the prompt text itself rather than from observed production outputs. We treat this as a starting point, not an endpoint. The registry is designed to accept additional behavioral dimensions without schema upheaval: the **health_analysis** column is JSON-typed, and a per-version **meta** blob carries extension fields.

Future Work

The registry described here is the persistence half of a fuller behavioral tracking system. Three extensions are planned and the current schema accommodates each without migration upheaval.

- **Sampled behavioral fingerprints.** Aggregating production samples per prompt version into a small fixed-shape record — refusal rate over the sampling window, output-length distribution percentiles,

embedding-based semantic consistency against a canonical input set — and storing the result in **health_analysis**.

- **Longitudinal comparison.** The Compare tab already supports text-level diff between any two versions; the same UI surface is intended to render side-by-side fingerprint deltas once the fingerprint record exists.
- **External event correlation.** Provider-side model updates and retrieval-corpus changes can be recorded as separate events in the version timeline so that observed behavioral shifts can be visually correlated with their candidate causes.

We name these explicitly because the registry's value rests on what is eventually built on top of it. Treating the prompt artifact as a versioned, provenance-bearing object is necessary but not sufficient; the analytical work that justifies the persistence happens in the layers above.

Status

The registry described in this note is implemented and in production use. The behavioral fingerprint extension and external-event correlation described under Future Work are design intent, not shipped functionality, at the time of writing.