Contextual AI-Driven Test Optimization Using Symbolic Hash Referencing

Author: James Gusty

Date: March 26, 2025

---

## Abstract

This document outlines an approach to optimizing test coverage within constrained AI systems by utilizing contextual identity referencing through cryptographic hashing of a comprehensive data dictionary. This method enables lightweight, contextually aware test generation that avoids over-validation of stable system components, allowing for precision-focused test coverage in high-volume environments.

## 1. Problem Statement

Our current testing methodology involves full-system validation for each programmatic implementation within our application. This leads to inefficient resource usage, redundant test execution, and difficulty scaling intelligent test coverage due to licensing and infrastructure constraints. These constraints limit our AI tooling to low token capacity and prevent persistent memory retention between prompts. We are constrained by LLM context limits and lack of persistent memory in our AI tooling.

## 2. Objective

Develop a contextual, symbolically compressed testing framework that:

- Avoids redundant baseline QA.

- Targets only program-specific transformations.

- Leverages cryptographic hashing as symbolic anchors.

- Enables AI-assisted prompt generation within token limits.

## 3. Core Concepts

## 3.1. Symbolic Compression through Hashing

Each table or entity in the data dictionary is hashed (e.g., SHA-256) to generate a lightweight, immutable reference. These hashes act as persistent symbolic identifiers.

## 3.2. Persistent Reference Layer

A lookup layer maps each hash to its corresponding full schema/table definition, which remains external to the AI context. This serves as a form of symbolic memory externalization.

## 3.3. Contextual Prompt Expansion

LLM input prompts consist of:

- The hashed reference(s) required.

- A user story or testing objective.

The LLM interprets the hashes as symbolic anchors, generating SQL validation or pseudocode by referencing schema logic stored externally and linked contextually.

## 3.4. Targeted Testing Logic

Instead of validating the full application baseline, only the affected program (or delta) is tested. This allows for focused regression coverage and more efficient QA workflows.

## 4. Implementation Roadmap

## Step 1: Data Dictionary Hashing

- Hash all schema/table definitions in the 1000+ page data dictionary.

- Store mappings in a secure reference index.

## Step 2: AI Prompt Framework Design

- Build prompt templates that accept (Hash + User Story).

- Return SQL scripts or assertions scoped only to the referenced table logic.

## Step 3: Test Reduction Strategy

- Catalog baseline-verified components.

- Mark them as stable (no need for revalidation).

- Flag only modified programs as needing recursive test generation.

## Step 4: Integration with Current QA Flows

- Layer AI-generated tests into existing test plan repositories.

- Automate diff detection to trigger prompt generation dynamically.

## Step 5: Monitoring & Refinement

- Log hash-prompt-query generation.

- Gather false positives/negatives.

- Continuously refine prompt structure.

## 5. Benefits

- Major reduction in redundant tests.

- Faster iteration cycles.

- AI augmentation of human QA strategy.

- Scalable even within token-limited AI environments.

## 6. References

- Internal Data Dictionary (confidential)

## Appendix **\*\*Sample Prompt Format\*\***

Input:

- Hash: 0x2a34ff...

- User Story: “As a user, I want to validate Program X retrieves the correct payment summary from Table Y.”

\*\*Expected Output (from AI):\*\*

SQL Validation:

SELECT \* FROM payment\_summary WHERE program\_id = 'X';