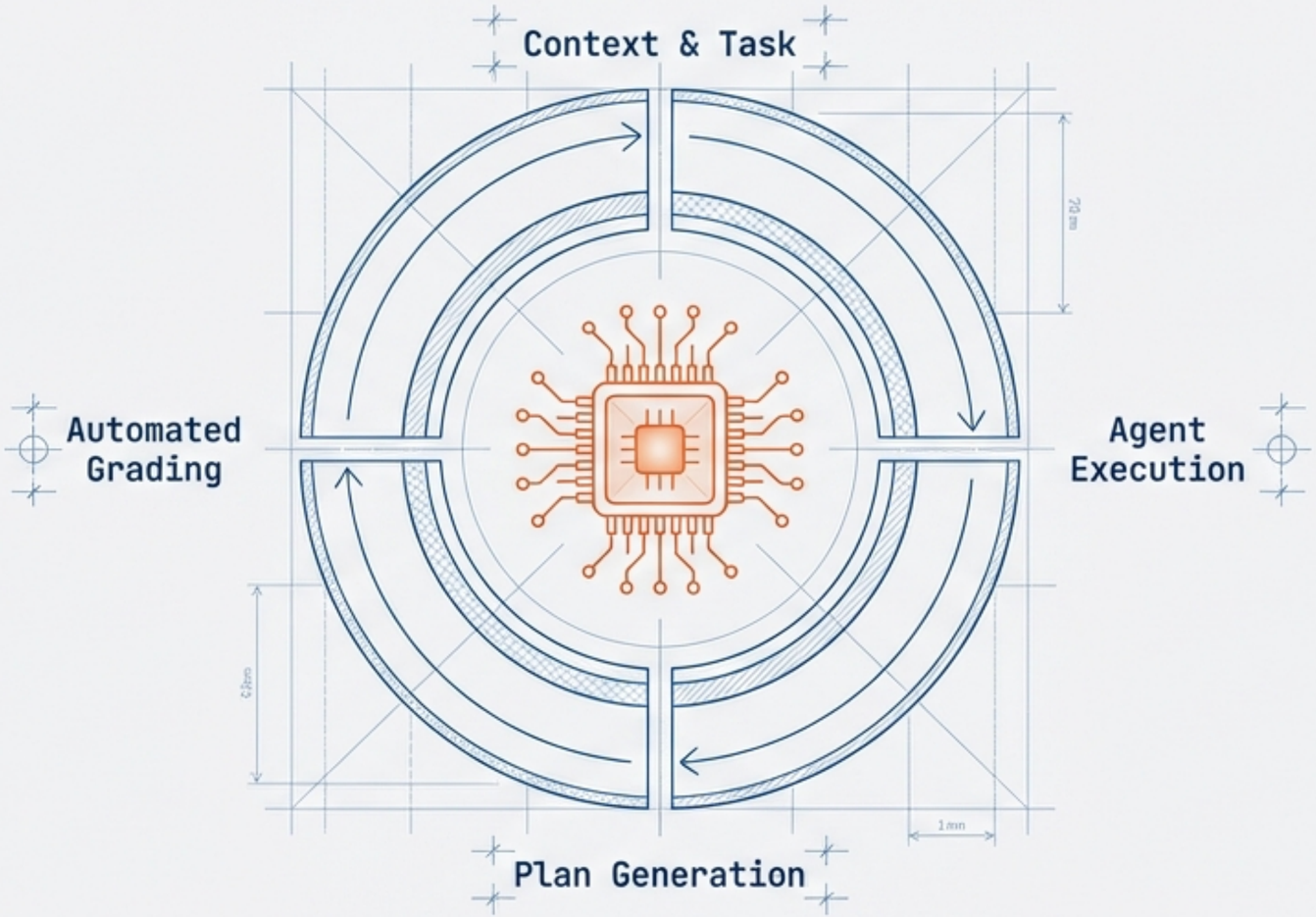
Planning Mode Eval

An Automated Evaluation Pipeline for Claude Code



A systems architecture walkthrough for measuring agentic planning capabilities at scale.

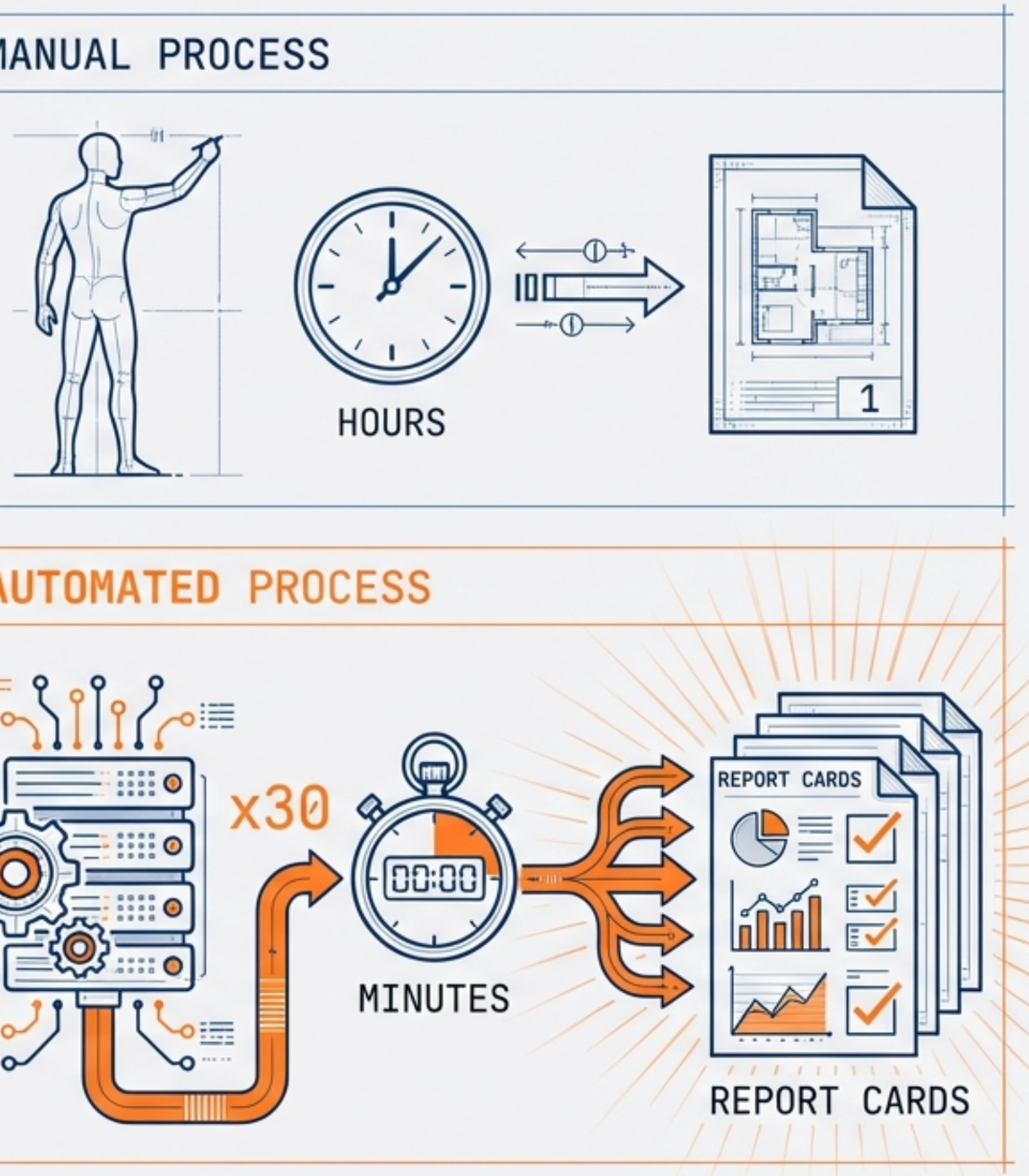
THE MISSION

OBJECTIVE

To rigorously test Claude Code's "Plan Mode" capabilities using an end-to-end automated pipeline.

KEY CHALLENGES

- **Scale:** Manual evaluation is unscalable. We need to process ~30 tasks per repository rapidly.
- **Autonomy:** The system must mimic a human developer's workflow—from understanding the codebase to validating the output—without human hand-holding.

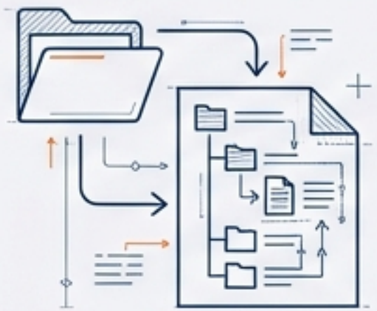


THE EVALUATION PIPELINE ARCHITECTURE

MODULE 0

CONTEXTIZE

Repo Map Creation



MODULE 1

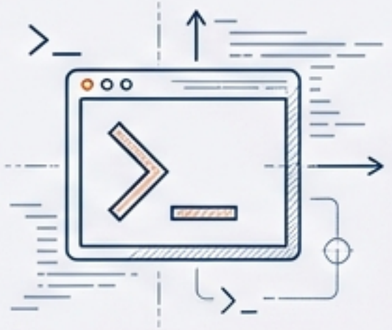
GENERATE TASKS

Repo Crawler

MODULE 2

RUN PLANS

Headless Execution



MODULE 3

GRADE

The Judge Model

INPUT: Git Repository

OUTPUT: JSON Scorecard

MODULE 0: CONTEXTIZING THE REPOSITORY

CONCEPT

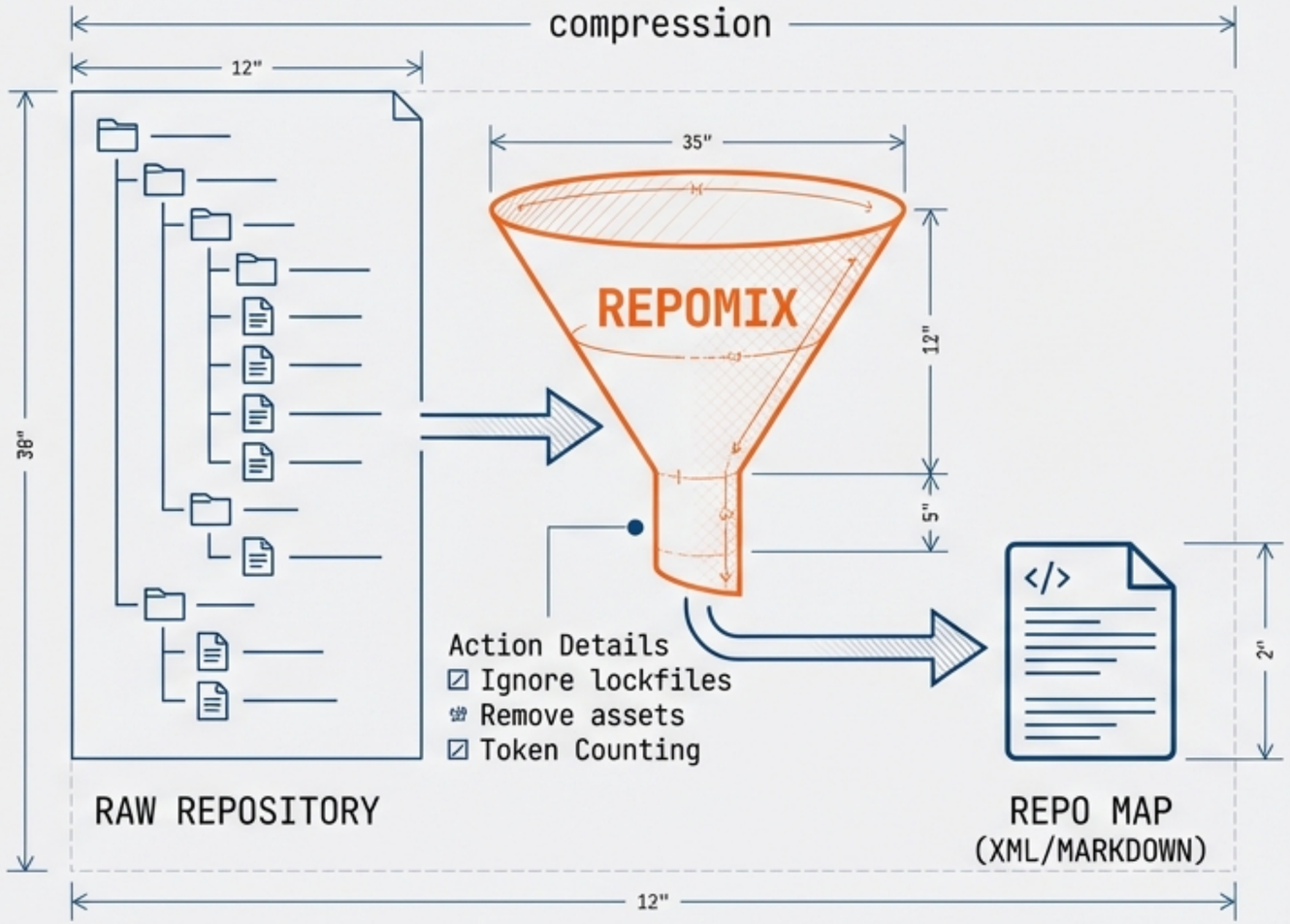
Transforming a directory of files into a single, high-density token stream.

THE TOOL: REPOMIX

Formerly Repopack. A dedicated tool for packing codebases for LLM consumption.

PROCESS

1. Clones the target repository.
2. Filters non-essential data.
3. Generates a compressed map fitting the context window.
4. Serves as ground truth for Agent and Judge.



MODULE 1: TASK GENERATION STRATEGY

We don't write tests; we mine them.

A. COMMIT REVERSE-ENGINEERING

Primary Method. Looking at the last 50-100 PRs/Commits to reconstruct the problem that led to the code change.

B. DOC-TO-TASK

Analyzing README.md or /docs to identify "Missing Features" or documented edge cases not yet implemented.

C. CODE SMELLS

Identifying high-complexity functions via cyclomatic complexity checks to generate refactoring tasks.

DEEP DIVE: REVERSE-ENGINEERING GIT HISTORY

AUTOMATION PIPELINE SCHEMATIC VOL. 02

MODULE 1

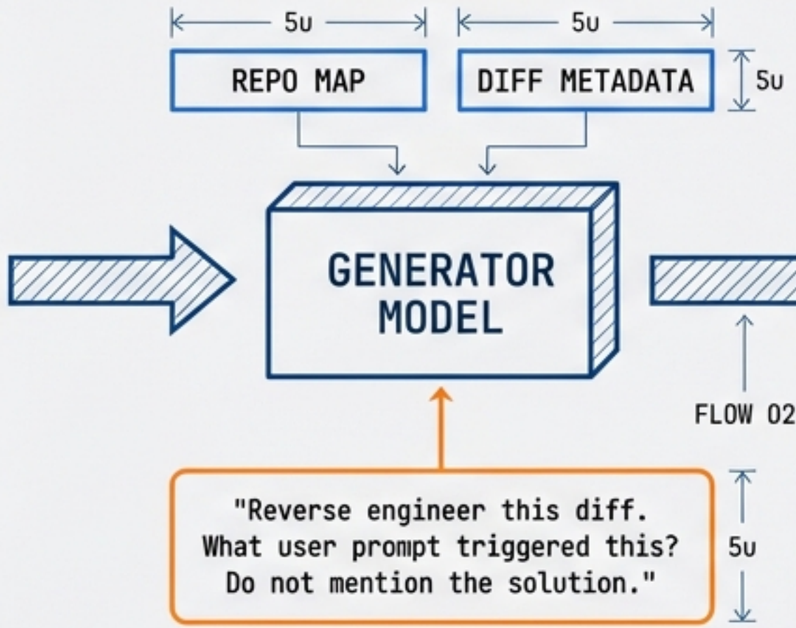
EXTRACTION

```
- function processData(input) {
-   return input.filter(i => i > 0);
- }
+ function processData(input, options) {
+   if (options.includeZero) {
+     return input.filter(i => i >= 0);
+   }
+   return input.filter(i => i > 0);
+ }
```

Pull Commit Metadata (Hash, Message, Diff)

MODULE 2

LLM TRANSFORMATION



MODULE 3

THE OUTPUT OBJECT

```
{
  "User Task (Synthetic Prompt)":
  "[Generated Prompt String]",

  "Repo State (Parent Commit)":
  "[Parent Hash/Context]",

  "Ground Truth (Actual PR Diff)":
  "[Diff Content]"
}
```

Take merge commits, and classify into:
(1) feature request (50%, 15/30)
(2) bug fix (30%, 9/30)
(3) code refactoring (20%, 6/30)
(4) do-not-use; i.e. not suitable for plan mode

MODULE 2: HEADLESS EXECUTION

Automation Pipeline Schematic Vol. 03

2. Execute Agent

use claude-agent-sdk

Automating the ~~interactive Claude Code CLI via ia wrapper scripts~~ to bypass the terminal UI.

1. State Reset

Clean slate. We revert the repo to the 'broken' or pre-feature state using the parent commit hash.

```
$ git checkout {parent_commit}
$ claude-code --headless --task '{user_task}'
$ write .claude/plan.md
```

3. Capture Plan

Intercepting standard output to save the Markdown plan for the Judge.

Timing: Asynchronous processing ~6 minutes per task.

MODULE 3: THE JUDGE MODEL

Model-Graded Evaluation using Claude ~~Opus~~

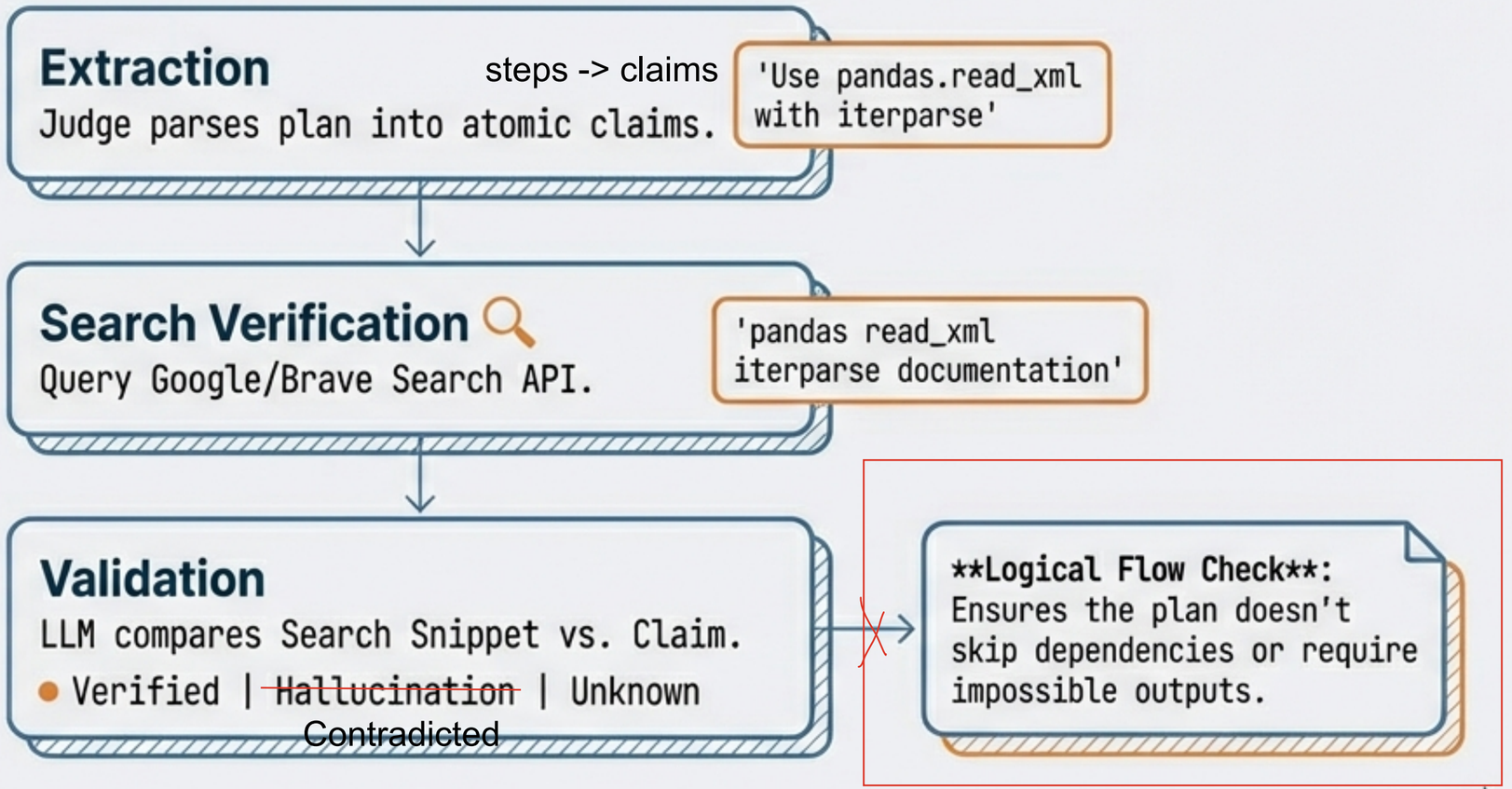
Opus 4.6 & Sonnet 4.5

NO Ground Truth

JetBrains Mono

WHY OPUS? A stronger model is required to audit the outputs of the smaller, faster agent model.

Grading Dimension A: Claim Verification (40%)



Logical soundness:
(1) Does any step require an output that a previous step fails to produce?
(2) Is the overall plan logically sound and does it solve the problem?

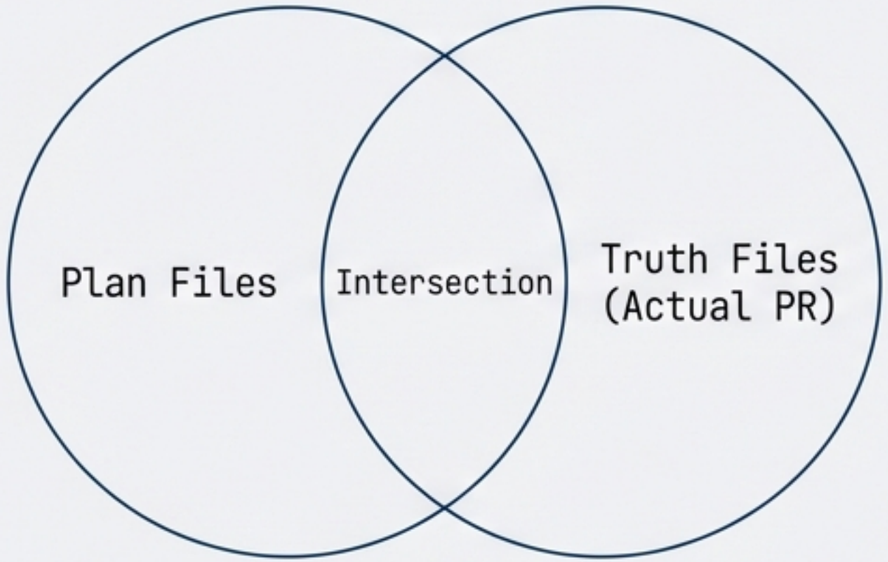
Opus 4.6

Timing: Processed as part of the Judge Model grading, ~1-2 minutes per task.

Grading Dimension B: Ground Truth Matching (40%)

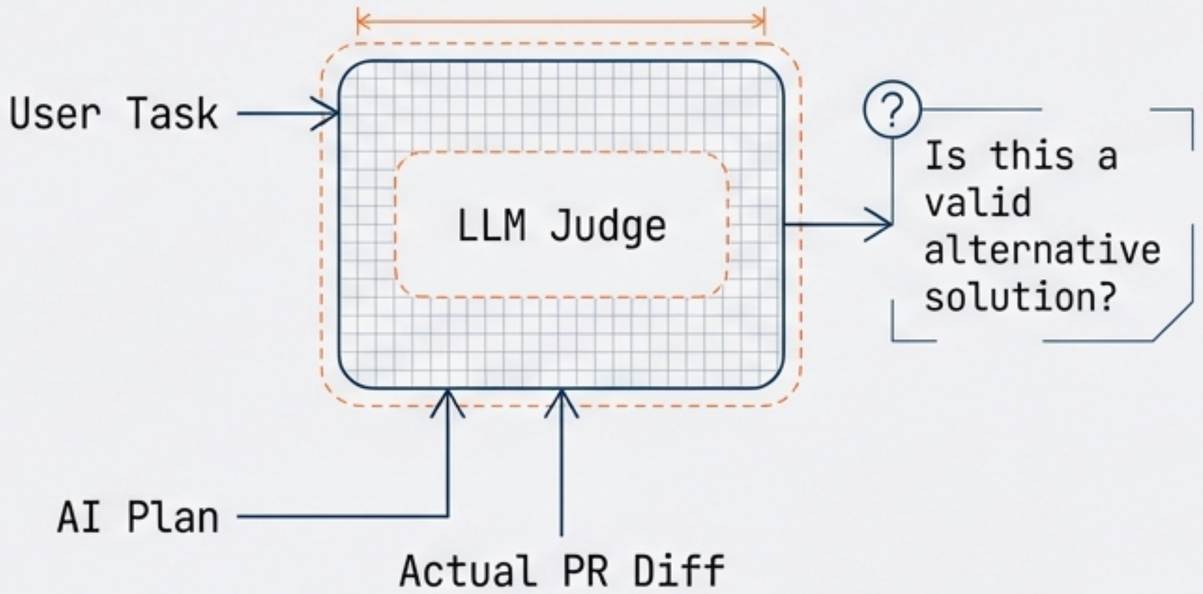
Opus 4.6

Metric 1: File Relevance (Static)



Recall = Intersection / Truth Files
Precision = Intersection / Plan Files

Metric 2: Semantic Equivalency (LLM Judge)



Grading Dimension C: Text Quality (20%)

The Style Guide Rubric

Conciseness

High information density. Low fluff.

Failure Mode Example:
“It is important to note that...”

Precision

Specific terminology over vague descriptions.

Failure Mode Example:
“Fix the thing...”

Tone

Professional, action-oriented.

Failure Mode Example:
“We could try to...”

Formatting

Proper markdown usage (bullets, code blocks).

Failure Mode Example:
“Wall of text”

The Final Scorecard

Task ID: 492 - Fix race condition in UserDB

- Claims Verification: **4/5 Verified** (Confirmed `asyncio.lock` usage)
- File Recall: **100%** (Correctly identified `user_db.py`)
- Style Score: **5/5** (Concise, Action-Oriented)

TOTAL SCORE: 92/100

The final output aggregates weighted sub-scores to provide a single signal of confidence for the plan.

Future Roadmap & Optimization

Track 1: Better Context

Current: Static Repo Maps → **Future:** Retrieval Augmented Generation (RAG) for large codebases.

Track 2: Better Tasks

Current: Heuristic Mining → **Future:** User Research driven tasks & LLM-refined ground truth labeling.

Current GT matching scores are low

Track 3: Better Grading

some claims are not directly verifiable with web searches

Current: Web Search → **Future:** Repo-aware verification & Human calibration.

Running the Pipeline

TERMINAL > src.run_pipeline

```
# 1. Install & Configure
pip install -r requirements.txt
cp .env.example .env # Set ANTHROPIC_API_KEY

# 2. Run Pipeline Steps
python -m src.run_pipeline contextize     # Build Map
python -m src.run_pipeline generate-tasks # Mine History
python -m src.run_pipeline run-plans      # Execute
python -m src.run_pipeline grade          # Score

# Or run all:
python -m src.run_pipeline all
```

Prerequisites:
Python, Node, Git, Claude Code CLI.

Technology Stack

Repomix

Context Engine

Claude Code SDK

Agent (Plan Mode)

Claude Opus

Judge Model

Search APIs

Google Custom Search / Brave

Python

Core Orchestration

Built for modularity and scalability.