CodeTaste: Can LLMs Generate Human-Level Code Refactorings?

Alex Thillen | Niels Mündler | Veselin Raychev | Martin Vechev

Paper & Code

Agents can execute well-specified refactorings, but struggle to discover human-like refactoring decisions autonomously.

CodeTaste evaluates alignment, behavior-preservation & human intent re-discovery of repository-wide refactorings.

Given only a high-level focus area, GPT-5.2 reaches 7.9% alignment; a separate planning stage nearly doubles this to 15.1%.

Background

Agents generate **working patches**, but code **quality erodes** over time.

Humans address this through **refactoring.**

Can coding agents come up with **human-like refactoring decisions** autonomously?

Two Tracks

Instructed Track
The **agent** receives a **highly detailed instruction** describing the desired **refactoring**.

Can coding agents **execute** repository-wide refactorings reliably?

Open Track
The **agent** only receives a vague **high-level focus area** for the refactoring.

Can coding agents **discover** human-like refactoring decisions autonomously?

Open Track Modes

Direct
The agent is instructed to perform a refactoring based on the focus area.

Plan
The agent is first instructed to concretize the focus area into a well-specified refactoring plan, then it has to implement this concrete plan.

Multiplan
The agent generates 5 distinct plans for the focus area. An Oracle selects the best-aligned plan. Then, the agent implements the best-aligned plan.

Metrics

The following example illustrates how we use static analysis to evaluate code patches. The additive rule $\gamma^+ \in \Gamma^+$ is satisfied by the 3 inserted lines in $\hat{X}$.

γ^+

```
pattern: $HOOK($STATE => $STATE.$FIELD)
metavariable-regex:
  metavariable: $HOOK
  regex: ^use[A-Z][a-zA-Z]*Store$
```

$\hat{X}$

```
@@ component excerpt @@
-  const { theme } = useStoreSelector(
       (state) => state.settings)
+  const theme = useSystemStore(
+     (state) => state.theme
+  )
```

We evaluate patch $\hat{X}$ applied on code-base R, denoted $R \circ \hat{X}$, using static analysis rules Γ and tests.

Instruction Following Rate Does the agent remove undesired patterns, measured by Γ^-, and introduce desired patterns Γ^+ ?

$$\textsc{Ifr}^+(\hat{X}) = \frac{1}{|\Gamma^+|} \sum_{\gamma \in \Gamma^+} \mathbb{1}_{\mathcal{M}}(\gamma, R \circ \hat{X})$$

$$\textsc{Ifr}^-(\hat{X}) = \frac{1}{|\Gamma^-|} \sum_{\gamma \in \Gamma^-} \left(1 - \mathbb{1}_{\mathcal{M}}(\gamma, R \circ \hat{X})\right)$$

$$\textsc{Ifr}(\hat{X}) = \frac{|\Gamma^-|}{|\Gamma|} \textsc{Ifr}^-(\hat{X}) + \frac{|\Gamma^+|}{|\Gamma|} \textsc{Ifr}^+(\hat{X})$$

Pass Is $R \circ \hat{X}$ a regression compared to reference state $R \circ X^*$ or initial state R?

$$\textsc{Pass}(\hat{X}) = \mathbb{1}\left[F_{R \circ \hat{X}} \leq f_{\max} \wedge P_{R \circ \hat{X}} \geq p_{\min}\right]$$

f_{max} and p_{min} are the maximum failing and minimum passing tests in $R \circ X^*$ and R.

Alignment Is the refactoring adhering to the refactoring **intent** and **behavior-preserving**?

$$\mathcal{A}(\hat{X}) = \textsc{Pass}(\hat{X}) \times \textsc{Ifr}(\hat{X})$$

Precision Did $\hat{X}$ avoid unrelated changes? Precision measures the share of edited lines that is involved in satisfying the static checks.

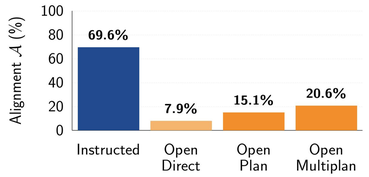
CodeTaste Pipeline

1. **Discover large refactoring commits**
2. **Generate task descriptions**
3. **Build reproducible environments**
4. **Generate static rules**
5. **Run coding agents**
6. **Evaluate with tests + rules**

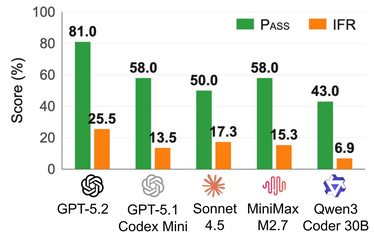
100	87	6	91.5	2,605	1,638	81
tasks	repositories	languages	files changed (avg)	lines edited (avg)	tests per task (avg)	static rules per task (avg)

Results

Execution vs. Discovery Gap
GPT-5.2 alignment on CodeTaste in different settings.



Open Track performance comparison in Multiplan Mode on Pass and IFR
GPT-5.2 most reliably aligns with the original refactoring intent, while preserving behavior.



Open Track Failure Modes

1. Fixate on unrelated salient subjects. Example: Agent is told to *improve internal organization and naming*, but only fixes class-name typos.
2. Workarounds
3. Destructive commands and leave the codebase in a broken state.
4. Focus on different parts of the system compared to the reference refactoring.

Related Work

Benchmark	# Lang	# Files edited	LoC edited	Intent	Tests	Static Checks
MiniCode	1	–	–	✗	✓	✗
Aider	1	1.0	–	✗	✗	✓
Can It Edit?	1	1.0	13.8	✗	✓	✗
RefactorMirror	1	1.0	25.3	✓	✓	✗
SWE-Refactor	1	1.1	52.2	✗	✓	✓
SWE-bench	1	1.7	32.8	✗	✓	✗
RefactorBench	1	4.3	68.3	✗	✗	✓
CodeTaste	**6**	**91.5**	**2605.4**	✓	✓	✓

CodeTaste uniquely combines repository-scale refactoring tasks with intent alignment validation, tests, and static checks.