> **TL;DR** We used SkillOpt as a starting point for improving Codex skills. The useful adaptation was simple: write synthetic edge cases, score the skill before changing it, make one narrow edit only when the miss is clear, then test on fresh unseen cases. A post-edit score tells us the fix did not break the old case. A first-unseen score tells us more about whether the skill is starting to generalize.

---

1 The problem: skills can look fine until the edge case arrives

Codex skills are small instruction files and helper scripts that steer an agent through a task. They are useful because they make repeated work less dependent on memory. They are risky for the same reason: once a skill sounds confident, it can hide the missing branch that only appears in an awkward real session.

That is easy to see with concrete examples:

- A diff-redaction skill can remove obvious API keys but accidentally destroy the file headers and hunk markers that make the diff useful.
- An MDX blockquote skill can catch a plain list marker but miss the same marker when it is indented or nested inside repeated quote markers.
- A session handoff skill can export the latest user turns but accidentally include injected context that should never be treated as user intent.
- A git restore skill can say it restores all uncommitted work but leave staged index changes behind.

These are not exotic failures. They are the kinds of misses that show up after enough users, repos, and agent sessions touch the same skill.

The question we wanted to answer was practical:

- Can we improve a skill without rewriting it from scratch?
- Can we tell the difference between a real general improvement and a fix that only matches yesterday's fixture?
- Can we keep the evidence small enough that future agents can inspect it?

2 What SkillOpt gave us

The SkillOpt paper treats an agent skill as something that can be optimized while the underlying model stays frozen. Its loop uses scored rollouts, bounded edits to a skill document, and held-out validation before accepting the edit.

The detail that mattered most for our work was not the exact optimizer setup. It was the discipline:

- do not rewrite the skill freely
- score behavior before editing
- make bounded changes
- accept changes only when held-out behavior improves

- preserve rejected edits and failures as evidence

We adapted that into a workflow that works for Codex skills and helper scripts. The agent can run the experiment itself, but the scoring rule has to be explicit enough that the agent cannot quietly count a patched case as fresh evidence.

3 Our adapted loop

The loop we used looked like this:

skill instructions -> synthetic OOD cases -> baseline score -> one narrow edit -> validation -> fresh OOD streak -> evidence report

Each part has a job.

The synthetic OOD cases are not meant to replace real usage. They are a first stress test for rare but important cases: staged git changes, odd MDX blockquote shapes, safe public config that looks secret-adjacent, or transcript wrappers that should be filtered out.

The baseline score happens before any edit. That protects us from the easiest self-deception: inventing a fix and then only testing the case the fix was written for.

The edit has to be narrow. If the miss is "staged changes are not restored", the edit should add staged-aware restore guidance. It should not turn the skill into a broad git tutorial.

The validation score checks that the edit fixed the known miss and did not regress older cases. That is useful, but it is not the main generalization signal.

The fresh OOD streak is the stronger test. We generated new unseen datasets after the edit and asked whether the skill still scored highly. For these runs, our rule was three consecutive valid first-unseen scores of at least 10 out of 12.

4 The scoring rule that kept us honest

The most important measurement decision was this:

First-unseen scores count as generalization evidence. Post-edit scores count as regression evidence.

That distinction changed how we interpreted the results.

If a skill scores 8 out of 12 on a new dataset, then we edit it and it scores 12 out of 12, that is a useful fix. But the 12 out of 12 does not prove the skill generalized to that dataset. The skill had already seen the failure through the scorer.

The next fresh dataset is the cleaner signal. If the skill keeps scoring 10, 11, or 12 out of 12 on new datasets that were written after the edit, we have better evidence that the skill improved beyond the original miss.

We also learned a benchmark-governance rule:

Invalid OOD rounds should be preserved, but excluded.

In the codex-session-handoff work, one OOD round contained an invalid assertion about how --last 3 should behave. That dataset still taught us something. It found real gaps. But it could not fairly count toward the streak or the final accumulated gate.

That matters because benchmark quality is part of the system. A bad test can make a good skill look wrong or make a weak skill look better than it is.

5 What changed across the skills

Here is the compact version of the evidence. The numbers below come from the OpenClaw SkillOpt reports, not from memory.

Skill	What the test caught	Accepted edit theme	First-unseen streak
redacted-diff	Public metadata and secret-looking structure were easy to over-redact or under-redact.	Preserve useful diff structure and public metadata while still redacting secret-bearing values.	OOD23 12 / 12, OOD24 12 / 12, OOD25 11 / 12
eclat-mdx-blockquote-safety	OOD1 scored 8 / 12 because the skill missed indented and nested blockquote marker shapes.	Generalize audit regexes across indentation and repeated quote markers.	OOD2 12 / 12, OOD3 12 / 12, OOD4 12 / 12
codex-session-handoff	The helper could leak injected context or let older real turns re-enter after filtering.	Skip injected wrappers and apply the tail window before filtering.	OOD3 11 / 12, OOD4 12 / 12, OOD5 11 / 12
restore-changes	OOD1 baseline scored 10 / 12 because staged changes were not handled fully.	Add staged-aware preview and restore guidance.	OOD2 12 / 12, OOD3 12 / 12, OOD4 12 / 12

The individual fixes were small. That was the point. The loop was useful because it pushed each skill toward a broader behavior without inviting a broad rewrite.

6 What the tests caught that casual use might miss

The strongest cases were not the obvious ones. They were edge cases that an agent might not meet for weeks, but that would eventually matter.

For redacted-diff, the hard part was not "redact passwords". It was preserving enough of the diff for a teammate to understand what changed while removing raw secret values. The final helper had to preserve headers, hunk markers, key names, authorization schemes, public hashes, and safe metadata while still blocking values that should not be shared.

For eclat-mdx-blockquote-safety, the value was in small syntax variations. A simple blockquote marker is easy. Indented markers, repeated quote markers, nested list markers, and trailing URL backslashes are more likely to slip through a human scan.

For codex-session-handoff, the important failure was transcript hygiene. The skill should carry useful context across compaction or cross-agent handoff. It should not treat injected environment blocks

or goal wrappers as user-authored material.

For restore-changes, the key miss was git index state. Restoring a worktree is not the same as unstaging and restoring the index. A skill that says "restore all uncommitted changes" has to make that distinction visible before any destructive action.

7 What this does not prove

This method does not prove that a skill is correct in the real world.

All of these OOD datasets were synthetic. That is useful because we can write rare edge cases before production exposes them. It is limited because synthetic generators can become stale, repetitive, or too close to the last failure.

The scores also do not prove that an agent will follow the skill perfectly under pressure. A future session can still misread instructions, skip validation, or obey a user request that should have triggered a safety gate.

The evidence is best read this way:

- A failed first-unseen score is a useful signal that the skill still has a gap.
- A post-edit pass is evidence that the specific fix works and old cases still pass.
- A run of fresh high first-unseen scores is bounded evidence that the edit generalized across the generated edge cases.
- Real user sessions remain the stronger long-term signal.

That is enough to make the method useful. It is not enough to stop watching the skill after it passes.

8 A reusable template for other skills

If you want to apply the same idea to another agent skill, keep the workflow small:

1. Pick one behavior. Do not test "make the skill better". Test something concrete, like "preserve diff structure while redacting secrets" or "do not include injected context in handoff exports".

2. Write 12 synthetic fixtures. Include true positives, false positives, and weird-but-real cases. Write the expected outcome before scoring.

3. Score before editing. Record the misses. If the baseline is already perfect, do not edit just to feel productive.

4. Make one bounded edit. Change the smallest instruction, regex, parser branch, or helper rule that explains the miss. Reject broad rewrites unless the scorer proves the skill shape itself is wrong.

5. Re-score the known dataset. Treat this as regression evidence, not fresh generalization evidence.

6. Generate a new unseen dataset. Score it before any edit for that dataset. This is the number that tells you whether the skill is starting to generalize.

7. Require a streak. Our default gate was three consecutive valid first-unseen scores of at least 10 out of 12. If a dataset has an invalid assertion, preserve it as evidence and exclude it from the streak.

8. Preserve the evidence. Keep the fixtures, scorer, results, accepted edit, rejected alternatives, and remaining risks where future agents can inspect them.

9 The broader lesson

The main lesson was not that synthetic tests are better than real usage. They are not.

The useful pattern is layered:

- Real sessions show what actually happens.
- Synthetic OOD cases test rare failures before they become production incidents.
- First-unseen scores help separate real generalization from patched-case success.
- Small evidence-backed edits keep the skill maintainable.
- Reports and mirrors keep the improvement from disappearing when an unversioned skill drifts.

That is the part worth keeping.

SkillOpt gave us a sharper way to think about skill improvement. The practical adaptation was to turn that idea into an operator loop: write edge cases, score honestly, edit narrowly, test fresh, and preserve the evidence.

For agent teams, that is a better default than waiting for a skill to fail in the middle of a real task.