

Evaluation of Code Generated by LLM-Based Coding Agents

Ing. Matěj Klouček · Supervisor: Ing. Jan Šedivý, CSc.

Master's thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2026



Motivation

AI coding agents can now build and modify whole web applications, but they often hand in code that *looks* finished and still does not work. A human developer would open the app and click through it before submitting. Can an agent do the same?

This work was carried out within the **CIIRC CTU** team in the **Amazon Nova AI Challenge 2026**, which develops a coding agent for full-stack web applications. The goal: find an architecture that lets the agent **verify its own end-to-end changes** without adding excessive overhead.

Three agent configurations

1. **Baseline**: the coding agent explores the repository, implements the task and opens a pull request, with no internal evaluation.
2. **Self-evaluation**: before coding, the agent writes its own Playwright end-to-end tests from the task description, then iterates until they pass.
3. **Agentic**: a separate evaluator agent tests the running application in a real browser (Playwright MCP), like a human tester, and reports back.

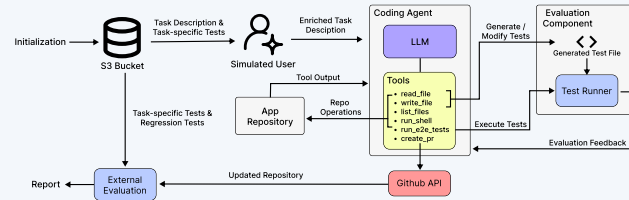
In both evaluation modes the agent may only submit once evaluation passes or after at most 5 fix-evaluate cycles.

Experimental setup

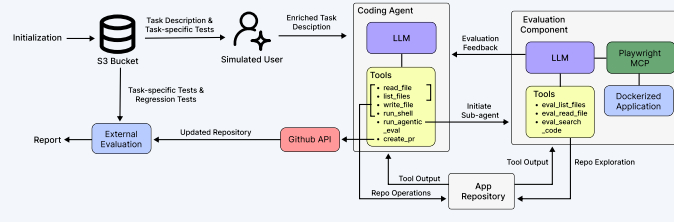
- **20** generated full-stack web apps (Python + JavaScript), **10 tasks** each (easy / medium / hard)
- **5 language models**: Nova 2 Lite, Qwen3 Coder 30B, MiniMax M2.5, GLM 4.7, Kimi K2.5
- $20 \times 10 \times 5 \times 3 = \mathbf{3,000}$ runs, executed in parallel on AWS
- Solutions graded by **hidden** end-to-end tests the agent never sees:
 - *Functional correctness*: Pass@1 on task tests
 - *Regression safety*: existing features still work
 - *Efficiency*: runtime and token consumption

Evaluation-enhanced architectures

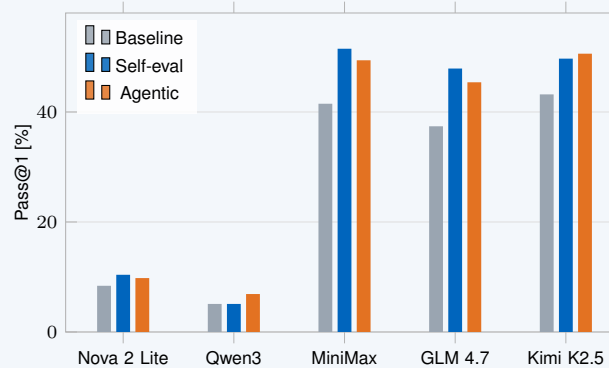
Self-evaluation: the agent writes the tests, runs them in Docker, reads the results.



Agentic: a separate evaluator agent clicks through the app in a browser. It never sees the code changes, so it is not biased by the implementation.



Results: functional correctness



- Overall Pass@1: **26.8%** → **31.6%** (self-eval) / **31.5%** (agentic)
- Stronger models gain **+6.5 to +10.5 pp**; weaker models barely improve

Why don't weaker models benefit?

A follow-up **cross-model experiment** swaps the coder and evaluator roles between a strong (Kimi K2.5) and a weak (Nova 2 Lite) model.

Coder	Evaluator	Pass@1	Δ vs. no eval
Kimi	Kimi	50.6%	+7.4 pp
Kimi	Nova	50.9%	+7.7 pp
Nova	Nova	9.8%	+1.4 pp
Nova	Kimi	19.0%	+10.6 pp

Paired with a strong evaluator, the weak coder **more than doubles** its success rate. Weak models *can* act on feedback; the bottleneck is the **quality of the feedback** they produce as evaluators.

Trade-offs

- **Runtime**: about 2× (self-eval) and 3× (agentic) the baseline; tokens about 2× for both
- **Regression safety**: preserved for strong models (>90%), but inaccurate feedback can push weak coders into breaking existing features
- **Complementary failure modes**: self-eval is too strict (70% precision, 23% recall), agentic too lenient (46% precision, 63% recall)

Impact

- Letting an AI agent **test its own work** yields more working code, as long as the evaluator is capable enough.
- Practical recipe: a **cheap model writes code, a strong model checks it**.
- Findings feed directly into the CIIRC CTU agent for the Amazon Nova AI Challenge.
- For everyday users: faster-built software with fewer bugs.