**Interview Transcript –** Participant: [Anonymized]

**Date of Interview:** 10 February 2025

**Mode of Interview:** Physical

**Interviewer:** Anonymized

**Participant Role:** Software Engineer (Team Lead)

**Duration:** 1 hour

**Consent Obtained:** Yes

**Interviewer:** Hi there, thanks for joining. **Participant:** Hello! Glad to be here.

**Interviewer:** I really appreciate you taking the time to participate in this conversation. We're going to talk a bit about code cloning today, particularly looking at the differences between AI-generated code and code written by humans. Before we get into that, could you briefly share your background and how long you’ve been working in software development?

**Participant:** Sure. I’ve been working in software development for over 7 years now with more than 10 years of programming experience. Currently, I’m a team lead at \*\*\*\* company.

**Interviewer:** Great, that gives some helpful context. Are you familiar with AI tools like ChatGPT, Gemini, Claude, and others?

**Participant:** Yes, I’ve used or explored most of them. I stay pretty up-to-date with those platforms.

**Interviewer:** So that means in your day-to-day development phase, you guys use any AI-assisted tools to write your code?

**Participant:** Yes, in our company, we do promote AI-assisted coding tools like Claude. It increases productivity.

**Interviewer**: Great to hear that, but are you familiar with code clones?

**Participant**: Yes, I’m very familiar with code clones. As part of my role, I conduct code reviews on a daily basis, and I often rely on clone detection tools. Identifying code clones is crucial, as they can introduce bugs or lead to maintenance issues in our software systems.

**Interviewer:** What do you think about the tendency of LLMS to generate cloned codes?

**Participant:** Actually, as a very heavy user, I can confirm that LLMs follow a structured coding pattern, and yes, sometimes it does give the same solution multiple times

**Interviewer:** Could you explain how your team checks for code that looks “copied” or overly similar?

**Participant:** Of course. We run two automatic checks each time someone tries to merge new code: Quick scan (A fast tool looks for obvious copy‑and‑paste situations) and Deeper scan (A second tool compares the new code against the rest of our project to catch less‑obvious similarities before anything is approved).

**Interviewer:** What changed after your team started using AI helpers?

**Participant:** The quick scan began missing more “look‑alike” pieces. For example, before AI we caught about 8 out of 10 clones; with AI code in the mix we dropped to roughly 4 out of 10.

**Interviewer:** Any idea why that happens?

**Participant:** Few main reasons: Extra comments and the old tools weren’t built to ignore so many comments. Polished renaming: Instead of quirky variable names like tmp or foo, AI sticks to neat, descriptive ones. Two pieces of code that do the exact same thing can look unrelated because the words are so tidy and consistent. Order shuffling: To sound fresh, the model may flip the order of helper steps, log first, then check errors, instead of the other way around. Humans spot that it’s still the same routine; the scanner, which expects a fixed sequence, misses the match.

**Interviewer:** Have you tried anything to fix that?

**Participant:** Yes. First, we told the quick scan to ignore all those extra comments. That nudged our hit rate back up from 4 out of 10 clones caught to about 5 out of 10. But that still means we’re missing roughly 50% of duplicates that slip in only because they come from AI assistants. We’re seeing false negatives in our clone detection pipeline that weren’t there before our team started using AI assistants. It’s a strong signal that the current detectors need more than a quick tweak; they need a real overhaul to understand how AI code hides its similarities.

**Interviewer:** Some people think AI mostly creates one kind of copy. What different kinds have you actually seen?

**Participant:** We’ve observed all sorts. Some cases: Same code, different variable names, some case: Same idea, different layout and some cases: Exact repeat. These variations produce a blind spot the older pipeline just wasn’t designed for.

**Interviewer:** Given that, what are your next steps to make sure clones don’t sneak through?

**Participant:** Feeding the detector a data of human and AI examples, so it learns the new patterns instead of guessing. Until we do that, false negatives will keep popping up. Honestly right now, we're flying blind when it comes to understanding how reliable our existing clone detection infrastructure is in this new development paradigm.

**Interviewer:** Do you still see an upside once the tools finally catch up?

**Participant:** Absolutely. Once detectors are updated, we can actually spot problems faster and with fewer false alarms than before. But we have to close the gap first.

**Demonstration Phase – Summary**

During this phase, the interviewer presented participants with several code snippets: one set written by humans (sourced from Stack Overflow) and another set generated by AI tools, both designed to perform the same functionality. Participants were asked to review the snippets and identify which ones were AI-generated and which were human-written.

**Key Insights:**-The participant couldn’t determine all the AI-generated code. (after removal of comments)

The participant found that LLM-generated code has more thorough documentation and structural consistency

- Classical tools face reduced effectiveness, thus leading to an increase in false negatives.

- Participant also pointed out that AI systems can produce all types of clones depending on the prompt engineering and implementation context.

- Participant emphasized reassessing the traditional clone detection tools because of the code authorship patterns.