We Let Our Agent Hack the Kernel and It Only Panicked Twice!

Tal Zussman[1] **Jeremy Carin**[2]
[1]Department of Computer Science, Columbia University, New York, NY
[2]MIT CSAIL, Cambridge, MA

Keywords: Linux kernel, coding agents, kernel debugging

1. INTRODUCTION AND RELATED WORK

The Linux kernel is one of the most demanding environments for coding agents. The codebase is massive and deeply interconnected, correctness standards are extreme, and debugging is significantly more challenging than userspace code. The kernel community has published agent-facing tooling and governance for LLM-assisted development [3,4], but recent efforts have focused on patch review: Sashiko [5] applies LLMs to automated mailing-list patch review, while Meta has published kernel-specific agent skills, including Semcode for code indexing and review-guidance prompts [6, 7]. These tools address individual steps of the development process, but integration into end-to-end agent workflows remains open.

We present our experiences using coding agents to write the majority of code for two systems papers with significant changes to the Linux kernel. We describe the environment and tools that make this tractable, and share the insights that emerge from agent-driven development in this pathological environment. We open-source our tools here: `https://github.com/tzussman/kernel-dev-skill`.

2. MOTIVATION

We highlight two kernel properties that challenge agents.

Complexity and scale. Kernel logic is spread across interconnected subsystems, and the volume of code exceeds LLM context windows. Understanding how a change in one subsystem affects another requires whole-system reasoning that extends beyond local code comprehension. Agents require strong search and retrieval abilities, combined with intuition on the design of architectures and how subsystems interact.

Verification and correctness. Agents work best with fast verification loops, where they modify code and test the changes. In contrast, the kernel requires a full build-and-boot cycle for every change. Bugs in the kernel are particularly difficult to solve: race conditions may never reproduce deterministically or surface only under specific workloads, and panics at boot can destroy observability. Standard userspace debugging tools do not exist in-kernel; tools like crash and KGDB require domain knowledge that LLMs lack. To prevent critical kernel bugs that lead to data corruption or security vulnerabilities, agents must statically and dynamically test correctness across a large search space.

3. ENVIRONMENT

We use Claude Code or Codex [2, 8] on the host with SSH access to a virtme-ng VM [9]. The agent builds the kernel on the host and boots it on the VM, reading serial port output to detect kernel panics, enabling iterative debugging. Development work happens on the host; debugging and testing happens in the VM over SSH. When SSHing into the VM, the agent loses its native tool calls and falls back to raw shell commands. We provide a set of skills to the agent to facilitate this environment.

VM. virtme-ng handles kernel configuration, compilation, and boot into a QEMU/KVM VM via MCP. We extend the MCP server with a skill to support reading from the serial port.

Code indexing. Semcode [7] provides Tree-sitter-based cross-referencing, call graph analysis, and lore archive search via MCP. Additionally, the agent writes skills specific to each subsystem that provide an index of the architecture.

Debugging. Custom skills wrap KGDB, crash, drgn, and bpftrace [1, 10–12]. For KGDB, the agent runs GDB on the host in a tmux session, connecting to the VM's KGDB stub over serial to set breakpoints and step through kernel execution interactively. For crash and drgn, the agent SSHs into the VM to analyze crash dumps and inspect live kernel state.

Review guidance. Review prompts [6] describe subsystem-specific review checklists that the agent applies to its changes.

Git. The agents tend to run into issues when rebasing and bisecting. A skill detailing pitfalls resolves this.

4. EXPERIMENT: SEMCODE ABLATION

We measure the impact of Semcode [7] on agent performance. We use Claude Opus 4.6 under three conditions: (A) kernel skill + Semcode MCP, (B) kernel skill only, (C) bare model. We test two tasks on v6.17-rc7: (T1) enumerate all callers of `folio_mark_dirty()`, (T2) find functions missed by a refactor. We find that the kernel skill yields more thorough yet significantly longer analysis, and using Semcode maintains that quality at lower cost. Additionally, both non-Semcode T1 tests identified multiple false positives or hallucinated.

	A (full)			B (skill)			C (bare)		
	Time	Cost	Pass	Time	Cost	Pass	Time	Cost	Pass
T1	61s	$0.17	✓	251s	$2.36	×	57s	$0.25	×
T2	69s	$0.26	✓	85s	$0.27	✓	83s	$0.34	✓

5. CONCLUSION

We find that general-purpose coding agents can produce *functional* code even for complex systems when given a well-designed environment and domain-specific tools. We believe that agents can serve as significant accelerants, yet are currently bounded by the underlying foundations of the codebase, and the capabilities and ingenuity of the humans driving them.

References

[1] D. Anderson. crash: Linux kernel crash dump analysis utility, 2022.

[2] Anthropic. Claude code: Anthropic's agentic coding tool, 2025.

[3] J. Corbet. On the use of LLM assistants for kernel development. *LWN.net*, Aug. 2025.

[4] S. Levin. AI coding assistants. Linux kernel documentation, `Documentation/process/coding-assistants.rst`, 2025.

[5] Linux Foundation. Sashiko: AI-powered patch review for the Linux kernel, 2025.

[6] C. Mason. Review prompts: AI code review prompts for the Linux kernel, 2025.

[7] Meta. Semcode: Kernel code indexing and MCP server, 2025.

[8] OpenAI. Codex: OpenAI's coding agent, 2025.

[9] A. Righi et al. virtme-ng: Kernel development and testing made easy, 2024.

[10] A. Robertson et al. bpftrace: High-level tracing language for Linux. `https://github.com/bpftrace/bpftrace`, 2018. Accessed: 2026.

[11] O. Sandoval. drgn: Programmable debugger for the Linux kernel, 2024.

[12] J. Wessel. Using kgdb, kdb and the kernel debugger internals. The Linux Kernel Documentation. Accessed: 2026-04-06.