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

Tal Zussman [1] Jeremy Carin [2]

[1]Columbia University [2]MIT CSAIL

Key Finding

With the right environment and tools, general-purpose coding agents can produce **functional operating system kernel code**.

Working in the Kernel Is Hard for Agents

The Linux kernel breaks the typical agent loop in two ways.

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.

Verification and correctness:

- Agents work best with fast verification loops, but the kernel requires a full build-and-boot cycle for every change.
- Bugs in the kernel can be particularly difficult: race conditions may not reproduce deterministically and panics can destroy observability.
- **Standard userspace debugging tools do not exist in-kernel**; tools like `crash` and KGDB require domain knowledge that LLMs lack.

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.

Related Work

The kernel community has published agent-based tooling for LLM-assisted development, but recent efforts focus on patch review:

- **Sashiko** applies LLMs to automated mailing-list patch review.
- **Semcode** is a code indexing database designed for the kernel.
- **Review prompts** provide subsystem-specific review checklists.

These tools address components of the development process, but **integration into end-to-end agent workflows remains open**.

Environment

We use Claude Code or Codex on the host with SSH access to a virtual machine managed by the agent. 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.

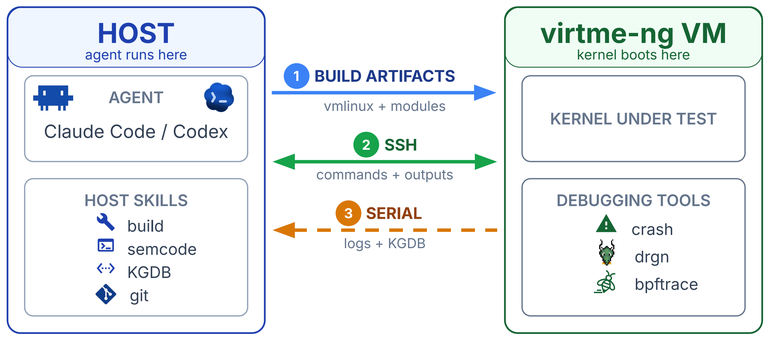

Figure 1. Agent operates on the host and SSHes into a VM for booting and debugging.

Skills

- **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 provides tree-sitter-based cross-referencing, call graph analysis, and mailing list archive search via MCP. The agent additionally writes skills specific to each subsystem that provide an index of the architecture.
- **Debugging.** Custom skills wrap KGDB, `crash`, drgn, and bpftrace. 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. For `crash` and drgn, the agent SSHs into the VM to analyze crash dumps and inspect live kernel state.
- **Review guidance.** Review prompts 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.

Experiment: Does Semcode Help?

We measure the impact of Semcode on agent performance, using Claude Opus 4.6 on Linux v6.17-rc7.

Tasks:

- **T1**: enumerate all callers of `folio_mark_dirty()`
- **T2**: find functions missed by a refactor

Conditions:

- **A**: kernel skill + Semcode MCP
- **B**: kernel skill only
- **C**: bare model

	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	✓

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.

Findings

- **Semcode's largest impact is on broad enumeration:** structured queries replace many grep+read cycles.
- **Tool impact is task-shape dependent:** file-scoped tasks see no meaningful Semcode advantage.

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.

When using this workflow for kernel development, we only ran into two kernel crashes when testing – a remarkably low number. The agents writing and reviewing code were able to catch or debug any issues before they reached human testing.

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.

Open-sourced at:
`github.com/tzussman/kernel-dev-skill`