Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

AI-Assisted Development with Rust

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- Professional Rust development experience (ownership, borrowing, traits, and error handling)
- Working knowledge of Git and GitHub (issues, pull requests, branches, reviews)
- Comfort with the command line and the cargo toolchain
- No prior AI tool experience required

Target Audience

Professional Rust developers who want a structured, end-to-end path from AI-curious to AI-proficient. Ideal for engineers and teams adopting AI coding tools across the full development lifecycle — writing, refactoring, testing, reviewing, and shipping Rust code — who also need to understand the security, licensing, and governance implications of doing so responsibly. This is a tool-agnostic course: students work hands-on with Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, and Gemini CLI rather than betting on a single vendor, and the skills transfer to whichever assistant your organization adopts.

Description

Rust’s compiler is the perfect partner for AI coding tools: generated code either satisfies the borrow checker or it doesn’t, and that fast, trustworthy feedback loop makes agentic workflows unusually effective. This five-day intensive teaches professional Rust developers how to exploit it. Days one and two cover the foundations: how large language models generate code, the strengths and tradeoffs of the frontier models on Rust specifically, and prompting and context engineering techniques that produce idiomatic, clippy-clean Rust rather than fighting the ownership model. Days two and three move into the tools: inline assistance and chat-driven editing in the IDE, then full agentic workflows across Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, and Gemini CLI — plan-then-execute sessions where the

agent compiles, reads compiler errors, and iterates; grounding agents with project context files; and issue-to-PR automation. Day four applies AI to quality: generating and maintaining test suites with cargo test and proptest, AI-assisted debugging of lifetime and trait-bound errors, code review, and incremental refactoring. Day five addresses team-scale adoption: reviewing AI-generated `unsafe` code, crate supply-chain vetting, licensing and IP questions, policy design, and honest productivity measurement. Students finish with a capstone that exercises the full workflow on a realistic Rust codebase.

Learning Outcomes

- Explain how LLMs generate code and assess frontier-model strengths and weaknesses on Rust.
- Apply prompting and context engineering techniques that produce idiomatic, clippy-clean Rust.
- Use IDE assistants (Copilot, Cursor, Windsurf) effectively alongside rust-analyzer for completion, chat, and multi-file edits.
- Drive agentic coding sessions with Claude Code, Copilot, Codex, Cursor, Windsurf, and Gemini CLI, using the compiler as the agent's feedback loop.
- Ground agents in project context with CLAUDE.md / AGENTS.md, rules files, and MCP servers.
- Generate, evaluate, and maintain test suites with cargo test, proptest, and AI assistance.
- Use AI to interpret and resolve borrow checker, lifetime, and trait-bound errors faster.
- Review AI-generated `unsafe` blocks, FFI boundaries, and concurrency code with appropriate rigor.
- Apply security guardrails: crate vetting with cargo-audit, secret handling, and prompt-injection awareness.
- Contribute to team standards for licensing, governance, and measuring AI's real productivity impact.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Software Requirements

Rust 1.93+ via rustup, a GitHub account with Copilot access, Claude Code CLI with an Anthropic API key or subscription, Codex, Cursor, Windsurf, and the Gemini CLI (trials and free tiers acceptable), VS Code with rust-analyzer, Git, and the GitHub CLI (`gh`).

Training Topics

Foundations: How AI Writes Code

- How LLMs generate code: tokens, context windows, and training data
- The frontier models for coding: Claude, GPT, and Gemini compared
- Why Rust's compiler makes agentic feedback loops unusually strong
- Where AI excels in Rust work and where it reliably fails

Prompting and Context Engineering for Rust

- Writing prompts that produce idiomatic Rust: ownership-aware design up front
- Steering toward proper error handling: `Result`, `thiserror`, and `anyhow`
- Context strategies: what to include from a workspace with many crates
- Trait design, generics, and lifetime annotations in prompts
- Avoiding outdated idioms and pre-2021-edition patterns in generated code

IDE Assistants in Daily Rust Work

- GitHub Copilot: completions, chat, edits, and Next Edit Suggestions
- Cursor: Composer, rules files, and project-level context
- Windsurf: Cascade, planning, and memories
- The wider landscape: Aider, Cline, JetBrains Junie, and Amazon Kiro
- Working with rust-analyzer and AI assistance together
- Multi-file edits and chat-driven refactoring across modules
- Doc comments, examples, and `cargo doc` generation

Agentic Coding: Claude Code, Copilot, Codex, Cursor, Windsurf, and Gemini CLI

- The agentic loop: plan, act, observe, iterate
- The agent landscape compared: Claude Code, Copilot agent mode, Codex, Cursor CLI, Windsurf Cascade, and Gemini CLI

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Plan mode and the supervision spectrum: permission prompts to auto mode
- CLAUDE.md and AGENTS.md: toolchain, workspace, and convention context
- The compile-fix loop: agents reading `cargo build` and clippy output
- Issue-to-PR workflows: assigning work to cloud agents
- MCP servers: connecting agents to internal tools and data

AI-Driven Testing and Quality

- Generating unit and integration tests for cargo test
- Property-based testing with proptest and AI assistance
- AI-assisted debugging: panics, lifetime errors, and trait-bound failures
- Agentic code review and pull request automation
- Incremental refactoring: characterization tests first, then change

Rust-Specific Pitfalls and Verification

- Reviewing AI-generated `unsafe` code and FFI boundaries
- Concurrency review: `Send`/`Sync` bounds and async pitfalls
- Hallucinated crates and API versions; verifying against docs.rs
- Clippy and rustfmt as quality gates for generated code
- Performance review: allocations, clones, and needless copies

Security, Licensing, and Governance

- Prompt injection and untrusted content in agent workflows
- Secret leakage: what never goes in a prompt or context file
- Crate supply-chain vetting: cargo-audit and cargo-deny
- Licensing and IP status of AI-generated code
- Team policy design: review requirements and audit trails
- Measuring real productivity impact honestly

Capstone Workshop

- Plan and execute a multi-step feature on a realistic Rust codebase with the coding agent of your choice
- Generate and harden a test suite for an untested module
- Issue-to-PR exercise: agent-ready issue through reviewed merge
- Security review of an AI-generated changeset including `unsafe` audit
- Q&A session