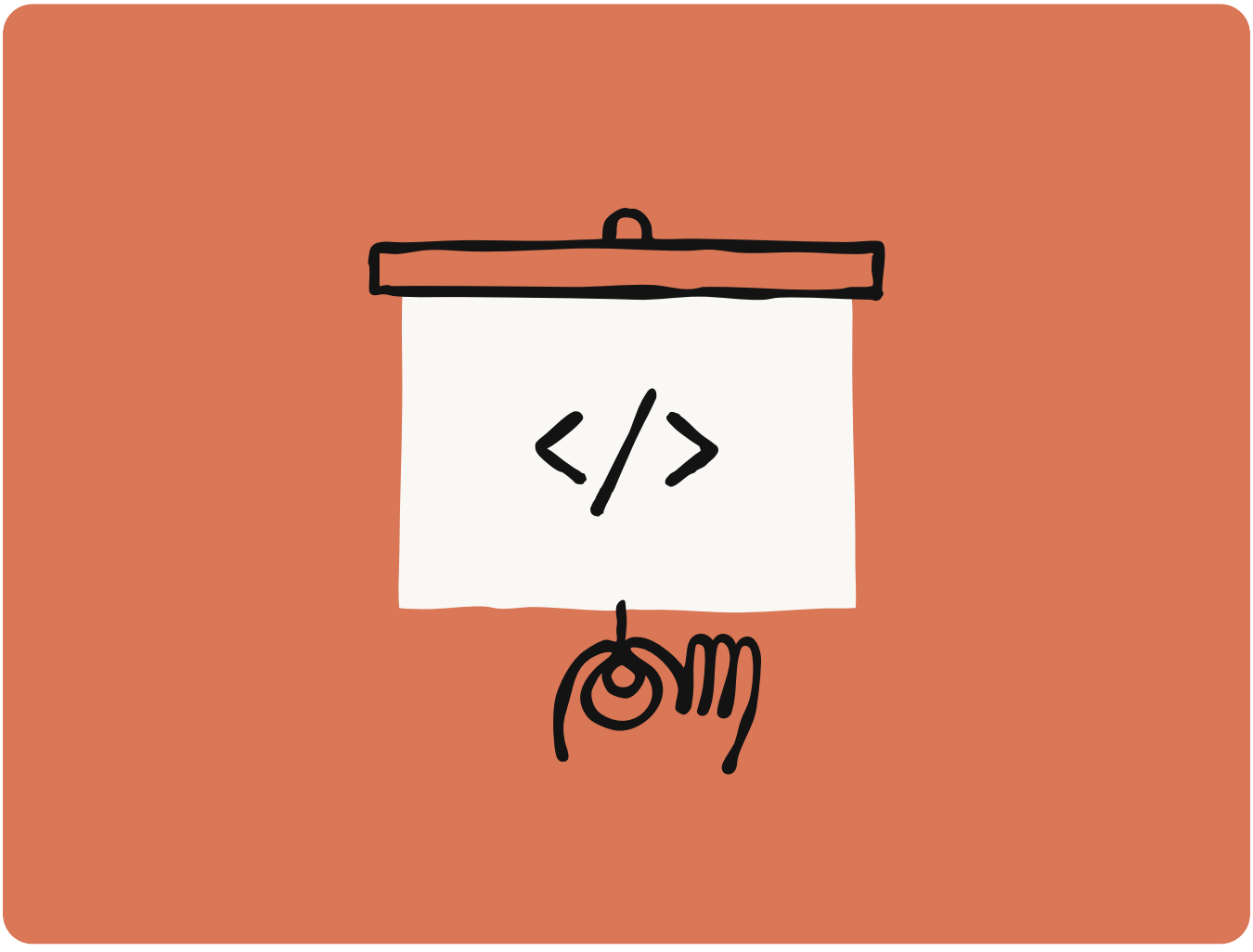
ANTHROP\C

How Anthropic teams use Claude Code



Anthropic's internal teams are transforming their workflows with Claude Code, enabling developers and non-technical staff to tackle complex projects, automate tasks, and bridge skill gaps that previously limited their productivity.

Through interviews with our own Claude Code power users, we've gathered insights on how different departments leverage Claude Code, its impact on their work, and tips for other organizations considering adoption.

Contents

Claude Code for data infrastructure

The Data Infrastructure team organizes all business data for teams across the company. They use Claude Code for automating routine data engineering tasks, troubleshooting complex infrastructure issues, and creating documented workflows for technical and non-technical team members to access and manipulate data independently.

Main Claude Code use cases

Kubernetes debugging with screenshots
When Kubernetes clusters went down and weren't scheduling new pods, the team used Claude Code to diagnose the issue. They fed screenshots of dashboards into Claude Code, which guided them through Google Cloud's UI menu by menu until they found a warning indicating pod IP address exhaustion. Claude Code then provided the exact commands to create a new IP pool and add it to the cluster, bypassing the need to involve networking specialists.

Plain text workflows for finance team
The team showed finance team members how to write plain text files describing their data workflows, then load them into Claude Code to get fully automated execution. Employees with no coding experience could describe steps like "query this dashboard, get information, run these queries, produce Excel output," and Claude Code would execute the entire workflow, including asking for required inputs like dates.

Codebase navigation for new hires
When new data scientists join the team, they're directed to use Claude Code to navigate their massive codebase. Claude Code reads their Claude.md files (documentation), identifies relevant files for specific tasks, explains data pipeline dependencies, and helps newcomers understand which upstream sources feed into dashboards. This replaces traditional data catalogs and discoverability tools.

End-of-session documentation updates
The team asks Claude Code to summarize completed work sessions and suggest improvements at the end of each task. This creates a continuous improvement loop where Claude Code helps refine the Claude.md documentation and workflow instructions based on actual usage, making subsequent iterations more effective.

Parallel task management across multiple instances
When working on long-running data tasks, they open multiple instances of Claude Code in different repositories for different projects. Each instance maintains full context, so when they switch back after hours or days, Claude Code remembers exactly what they were doing and where they left off, enabling true parallel workflow management without context loss.

Claude Code for data infrastructure

Team impact

Resolved infrastructure problems without specialized expertise
Resolved Kubernetes cluster issues that would normally require pulling in systems or networking team members, using Claude Code to diagnose problems and provide exact fixes.

Accelerated onboarding
New data analysts and team members can quickly understand complex systems and contribute meaningfully without extensive guidance.

Enhanced support workflow
Can process much larger data volumes and identify anomalies (like monitoring 200 dashboards) that would be impossible for humans to review manually.

Enabled cross-team self-service
Finance teams with no coding experience can now execute complex data workflows independently.

Top tips from the Data Infrastructure team

Write detailed Claude.md files
The better you document your workflows, tools, and expectations in Claude.md files, the better Claude Code performs. This made Claude Code excel at routine tasks like setting up new data pipelines when you have existing patterns.

Use MCP servers instead of CLI for sensitive data
They recommend using MCP servers rather than the BigQuery CLI to maintain better security control over what Claude Code can access, especially for handling sensitive data that requires logging or has potential privacy concerns.

Share team usage sessions
The team held sessions where members demonstrated their Claude Code workflows to each other. This helped spread best practices and showed different ways to use the tool they might not have discovered on their own.

Claude Code for product development

The Claude Code team uses their own product to build updates to Claude Code, expanding the product's enterprise capabilities and agentic loop functionalities.

Main Claude Code use cases

Fast prototyping with auto-accept mode
Engineers use Claude Code for rapid prototyping by enabling "auto-accept mode" (shift+tab) and setting up autonomous loops where Claude writes code, runs tests, and iterates continuously. They give Claude abstract problems they're unfamiliar with, let it work autonomously, then review the 80% complete solution before taking over for final refinements. Teams emphasize starting from a clean git state and committing checkpoints regularly so they can easily revert any incorrect changes if Claude goes off track.

Synchronous coding for core features
For more critical features touching the application's business logic, the team works synchronously with Claude Code, giving detailed prompts with specific implementation instructions. They monitor the process in real-time to ensure code quality, style guide compliance, and proper architecture while letting Claude handle the repetitive coding work.

Building Vim mode
One of their most successful async projects was implementing Vim key bindings for Claude Code. They asked Claude to build the entire feature (despite it not being a priority), and roughly 70% of the final implementation came from Claude's autonomous work, requiring only a few iterations to complete.

Test generation and bug fixes
They use Claude Code to write comprehensive tests after implementing features and handle simple bug fixes identified in pull request reviews. They also leverage GitHub Actions integration to have Claude automatically address Pull Request comments like formatting issues or function renaming.

Codebase exploration
When working with unfamiliar codebases (like the monorepo or API side), the team uses Claude Code to quickly understand how systems work. Instead of waiting for Slack responses, they ask Claude directly for explanations and code references, saving significant time in context switching.

Claude Code for product development

Team impact

Faster feature implementation
Successfully implemented complex features like Vim mode with 70% of code written autonomously by Claude.

Improved development velocity
Can rapidly prototype features and iterate on ideas without getting bogged down in implementation details.

Enhanced code quality through automated testing
Claude generates comprehensive tests and handles routine bug fixes, maintaining high standards while reducing manual effort.

Better codebase exploration
Team members can quickly understand unfamiliar parts of the monorepo without waiting for colleague responses.

Top tips from the Claude Code team

Create self-sufficient loops
Set up Claude to verify its own work by running builds, tests, and lints automatically. This allows Claude to work longer autonomously and catch its own mistakes, especially effective when you ask Claude to generate tests before writing code.

Develop task classification intuition
Learn to distinguish between tasks that work well asynchronously (peripheral features, prototyping) versus those needing synchronous supervision (core business logic, critical fixes). Abstract tasks on the product's edges can be handled with "auto-accept mode," while core functionality requires closer oversight.

Form clear, detailed prompts
When components have similar names or functions, be extremely specific in your requests. The better and more detailed your prompt, the more you can trust Claude to work independently without unexpected changes to the wrong parts of the codebase.

Claude Code for security engineering

The Security Engineering team focuses on securing the software development lifecycle, supply chain security, and development environment security. They use Claude Code extensively for writing and debugging code.

Main Claude Code use cases

Complex infrastructure debugging
When working on incidents, they feed Claude Code stack traces and documentation, asking it to trace control flow through the codebase. This significantly reduces time-to-resolution for production issues, allowing them to understand problems that would normally take 10-15 minutes of manual code scanning in about 5 minutes.

Terraform code review and analysis
For infrastructure changes requiring security approval, they copy Terraform plans into Claude Code to ask "what's this going to do? Am I going to regret this?" This creates tighter feedback loops and makes it easier for the security team to quickly review and approve infrastructure changes, reducing bottlenecks in the development process.

Documentation synthesis and runbooks
They have Claude Code ingest multiple documentation sources and create markdown runbooks, troubleshooting guides, and overviews. They use these condensed documents as context for debugging real issues, creating a more efficient workflow than searching through full knowledge bases.

Test-driven development workflow
Instead of their previous "design doc → janky code → refactor → give up on tests" pattern, they now ask Claude Code for pseudocode, guide it through test-driven development, and periodically check in to steer it when stuck, resulting in more reliable and testable code.

Context switching and project onboarding
When contributing to existing projects like "dependant" (a web application for security approval workflows), they use Claude Code to write, review, and execute specifications written in markdown and stored in the codebase, enabling meaningful contributions within days instead of weeks.

Claude Code for security engineering

Team impact

Reduced incident resolution time
Infrastructure debugging that normally takes 10-15 minutes of manual code scanning now takes about 5 minutes.

Improved security review cycle
Terraform code reviews for security approval happen much faster, eliminating developer blocks while waiting for security team approval.

Enhanced cross-functional contribution
Team members can meaningfully contribute to projects within days instead of weeks of context building.

Better documentation workflow
Synthesized troubleshooting guides and runbooks from multiple sources create more efficient debugging processes.

Top tips from the Security Engineering team

Use custom slash commands extensively
Security engineering uses 50% of all custom slash command implementations in the entire monorepo. These custom commands streamline specific workflows and speed up repeated tasks.

Let Claude talk first
Instead of asking targeted questions for code snippets, they now tell Claude Code to “commit your work as you go” and let it work autonomously with periodic check-ins, resulting in more comprehensive solutions.

Leverage it for documentation
Beyond coding, Claude Code excels at synthesizing documentation and creating structured outputs. They provide writing samples and formatting preferences to get documents they can immediately use in Slack, Google Docs, and other tools to avoid interface switching fatigue.

Claude Code for inference

The Inference team manages the memory system that stores information while Claude reads your prompt and generates its response. Team members, especially those who are new to machine learning, can use Claude Code extensively to bridge that knowledge gap and accelerate their work.

Main Claude Code use cases

Codebase comprehension and onboarding
The team relies heavily on Claude Code to quickly understand the architecture when joining a complex codebase. Instead of manually searching GitHub repos, they ask Claude to find which files call specific functionalities, getting results in seconds rather than asking colleagues or searching manually.

Unit test generation with edge case coverage
After writing core functionality, they ask Claude to write comprehensive unit tests. Claude automatically includes missed edge cases, completing what would normally take significant mental energy in minutes, acting like a coding assistant they can review.

Machine learning concept explanation
Without a machine learning background, team members depend on Claude to explain model-specific functions and settings. What would require an hour of Google searching and reading documentation now takes 10-20 minutes, reducing research time by 80%.

Cross-language code translation
When testing functionality in different programming languages, they explain what they want to test and Claude writes the logic in the required language (like Rust), eliminating the need to learn new languages just for testing purposes.

Command recall and Kubernetes management
Instead of remembering complex Kubernetes commands, they ask Claude for the correct syntax, like “how to get all pods or deployment status,” and receive the exact commands needed for their infrastructure work.

Claude Code for inference

Team impact

Accelerated ML concept learning
Research time reduced by 80% - what took an hour of Google searching now takes 10-20 minutes.

Faster codebase navigation
Can find relevant files and understand system architecture in seconds instead of asking colleagues.

Comprehensive test coverage
Claude automatically generates unit tests with edge cases, relieving mental burden while maintaining code quality.

Language barrier elimination
Can implement functionality in unfamiliar languages like Rust without needing to learn it.

Top tips from the Inference team

Test knowledge base functionality first
Try asking various questions to see if Claude can answer faster than Google search. If it's faster and more accurate, it's a valuable time-saving tool for your workflow.

Start with code generation
Give Claude specific instructions and ask it to write logic, then verify correctness. This helps build trust in the tool's capabilities before using it for more complex tasks.

Use it for test writing
Having Claude write unit tests relieves significant pressure from daily development work. Leverage this feature to maintain code quality without spending time thinking through all test cases manually.

Claude Code for data science and visualization

Data Science and ML Engineering teams need sophisticated visualization tools to understand model performance, but building these tools often requires expertise in unfamiliar languages and frameworks. Claude Code enables these teams to build production-quality analytics dashboards without becoming full-stack developers.

Main Claude Code use cases

Building JavaScript/TypeScript dashboard apps
Despite knowing "very little JavaScript and TypeScript," the team uses Claude Code to build entire React applications for visualizing RL model performance and training data. They give Claude control to write full applications from scratch, like a 5,000-line TypeScript app, without needing to understand the code themselves. This is critical because visualization apps are relatively low context and don't require understanding the entire monorepo, allowing rapid prototyping of tools to understand model performance during training and evaluations.

Handling repetitive refactoring tasks
When faced with merge conflicts or semi-complicated file refactoring that's too complex for editor macros but not large enough for major development effort, they use Claude Code like a "slot machine" - commit their state, let Claude work autonomously for 30 minutes, and either accept the solution or restart fresh if it doesn't work.

Creating persistent analytics tools instead of throwaway notebooks
Instead of building one-off Jupyter notebooks that get discarded, the team now has Claude build permanent React dashboards that can be reused across future model evaluations. This is important because understanding Claude's performance is "one of the most important things for the team" - they need to understand how models perform during training and evaluations, which "is actually non-trivial and simple tools can't get too much signal from looking at a single number go up."

Zero-dependency task delegation
For tasks in completely unfamiliar codebases or languages, they delegate entire implementation to Claude Code, leveraging its ability to gather context from the monorepo and execute tasks without their involvement in the actual coding process. This allows productivity in areas outside their expertise instead of spending time learning new technologies.

Claude Code for data science and visualization

Team impact

Achieved 2-4x time savings
Routine refactoring tasks that were tedious but manageable manually are now completed much faster.

Built complex applications in unfamiliar languages
Created 5,000-line TypeScript applications despite having minimal JavaScript/TypeScript experience.

Shifted from throwaway to persistent tools
Instead of disposable Jupyter notebooks, now building reusable React dashboards for model analysis.

Direct model improvement insights
Firsthand Claude Code experience informs development of better memory systems and UX improvements for future model iterations.

Enabled visualization-driven decision making
Better understanding of Claude's performance during training and evaluations through advanced data visualization tools.

Top tips from the Data Science and ML Engineering teams

Treat it like a slot machine
Save your state before letting Claude work, let it run for 30 minutes, then either accept the result or start fresh rather than trying to wrestle with corrections. Starting over often has a higher success rate than trying to fix Claude's mistakes.

Interrupt for simplicity when needed
While supervising, don't hesitate to stop Claude and ask "why are you doing this? Try something simpler." The model tends toward more complex solutions by default but responds well to requests for simpler approaches.

Claude Code for API

The API Knowledge team works on features like PDF support, citations, and web search that bring additional knowledge into Claude's context window. Working across large, complex codebases means constantly encountering unfamiliar code sections, spending significant time understanding which files to examine for any given task, and building context before making changes. Claude Code improves this experience by serving as a guide that can help them understand system architecture, identify relevant files, and explain complex interactions.

Main Claude Code use cases

First-step workflow planning
The team uses Claude Code as their "first stop" for any task, asking it to identify which files to examine for bug fixes, feature development, or analysis. This replaces the traditional time-consuming process of manually navigating the codebase and gathering context before starting work.

Independent debugging across codebases
The team now has the confidence to tackle bugs in unfamiliar parts of the codebase instead of asking others for help. They can ask Claude "Do you think you can fix this bug? This is the behavior I'm seeing" and often get immediate progress, which wasn't feasible before given the time investment required.

Model iteration testing through dogfooding
Claude Code automatically uses the latest research model snapshots, making it their primary way of experiencing model changes. This gives them direct feedback on model behavior changes during development cycles, which they hadn't experienced during previous launches.

Eliminating context-switching overhead
Instead of copying code snippets and dragging files into Claude.ai while explaining problems extensively, they can ask questions directly in Claude Code without additional context gathering, significantly reducing mental overhead.

Claude Code for API

Team impact

Increased confidence in tackling unfamiliar areas
Team members can independently debug bugs and investigate incidents in unfamiliar codebases.

Significant time savings in context gathering
Eliminated the overhead of copying code snippets and dragging files into Claude.ai, reducing mental context-switching burden.

Faster rotation onboarding
Engineers rotating to new teams can quickly navigate unfamiliar codebases and contribute meaningfully without extensive colleague consultation.

Enhanced developer happiness
Team reports feeling happier and more productive with reduced friction in daily workflows.

Top tips from the API Knowledge team

Treat it as an iterative partner, not a one-shot solution
Rather than expecting Claude to solve problems immediately, approach it as a collaborator you iterate with. This works better than trying to get perfect solutions on the first try.

Use it for building confidence in unfamiliar areas
Don’t hesitate to tackle bugs or investigate incidents outside your expertise - Claude Code makes it feasible to work independently in areas that would normally require extensive context building.

Start with minimal information
Begin with just the bare minimum of what you need and let Claude guide you through the process, rather than front-loading extensive explanations.

Claude Code for growth marketing

The Growth Marketing team focuses on building out performance marketing channels across paid search, paid social, mobile app stores, email marketing, and SEO. As a non-technical team of one, they use Claude Code to automate repetitive marketing tasks and create agentic workflows that would traditionally require significant engineering resources.

Main Claude Code use cases

Automated Google Ads creative generation
The team built an agentic workflow that processes CSV files containing hundreds of existing ads with performance metrics, identifies underperforming ads for iteration, and generates new variations that meet strict character limits (30 characters for headlines, 90 for descriptions). Using two specialized sub-agents (one for headlines, one for descriptions), the system can generate hundreds of new ads in minutes instead of requiring manual creation across multiple campaigns. This has enabled them to test and iterate at scale, something that would have taken a significant amount of time to achieve previously.

Figma plugin for mass creative production
Instead of manually duplicating and editing static images for paid social ads, they developed a Figma plugin that identifies frames and programmatically generates up to 100 ad variations by swapping headlines and descriptions, reducing what would take hours of copy-pasting to half a second per batch. This enables 10x creative output, allowing the team to test vastly more creative variations across key social channels.

Meta Ads MCP server for campaign analytics
They created an MCP server integrated with Meta Ads API to query campaign performance, spending data, and ad effectiveness directly within the Claude Desktop app, eliminating the need to switch between platforms for performance analysis, saving critical time where every efficiency gain translates to better ROI.

Advanced prompt engineering with memory systems
They implemented a rudimentary memory system that logs hypotheses and experiments across ad iterations, allowing the system to pull previous test results into context when generating new variations, creating a self-improving testing framework. This enables systematic experimentation that would be impossible to track manually.

Claude Code for growth marketing

Team impact

Dramatic time savings on repetitive tasks
Ad copy creation reduced from 2 hours to 15 minutes, freeing up time for strategic work.

10x increase in creative output
The team can now test vastly more ad variations across channels with automated generation and Figma integration.

Operating like a larger team
The team can handle tasks that traditionally required dedicated engineering resources.

Strategic focus shift
The team can spend more time on overall strategy and building agentic automation rather than manual execution.

Top tips from the Growth Marketing team

Identify API-enabled repetitive tasks
Look for workflows involving repetitive actions with tools that have APIs (like ad platforms, design tools, analytics platforms). These are prime candidates for automation and where Claude Code provides the most value.

Break complex workflows into specialized sub-agents
Instead of trying to handle everything in one prompt or workflow, create separate agents for specific tasks (like their headline agent vs. description agent). This makes debugging easier and improves output quality when dealing with complex requirements.

Thoroughly brainstorm and prompt plan before coding
Spend significant time upfront using Claude.ai to think through your entire workflow, then have Claude.ai create a comprehensive prompt and code structure for Claude Code to reference. Also, work step-by-step rather than asking for one-shot solutions to avoid Claude getting overwhelmed by complex tasks.

Claude Code for product design

The Product Design team supports Claude Code, Claude.ai and the Anthropic API, specializing in building AI products. Even non-developers can use Claude Code to bridge the traditional gap between design and engineering, enabling direct implementation of their design vision without extensive back-and-forth with engineers.

Main Claude Code use cases

Front-end polish and state management changes
Instead of creating extensive design documentation and going through multiple rounds of feedback with engineers for visual tweaks (typefaces, colors, spacing), they now directly implement these changes using Claude Code. Engineers noted they're making "large state management changes that you typically wouldn't see a designer making," enabling them to achieve the exact quality they envision.

GitHub Actions automated ticketing
Using Claude Code's GitHub integration, they can simply file issues/tickets describing needed changes, and Claude automatically proposes code solutions without having to open Claude Code, creating a seamless bug-fixing and feature refinement workflow for their persistent backlog of polish tasks.

Rapid interactive prototyping
By pasting mockup images into Claude Code, they generate fully functional prototypes that engineers can immediately understand and iterate on, replacing the traditional cycle of static Figma designs that required extensive explanation and translation to working code.

Edge case discovery and system architecture understanding
They use Claude Code to map out error states, logic flows, and different system statuses, allowing them to identify edge cases during design rather than discovering them later in development, fundamentally improving the quality of their initial designs.

Complex copy changes and legal compliance
For tasks like removing "research preview" messaging across the entire codebase, they used Claude Code to find all instances, review surrounding copy, coordinate changes with legal in real-time, and implement updates - a process that took two 30-minute calls instead of a week of back-and-forth coordination.

Claude Code for product design

Team impact

Transformed core workflow
Claude Code becomes a primary design tool, with Figma and Claude Code open 80% of the time.

2-3x faster execution
Visual and state management changes that previously required extensive back-and-forth with engineers now implemented directly.

Weeks to hours cycle time
Complex projects like GA launch messaging that would take a week of coordination now completed in two 30-minute calls.

Two distinct user experiences
Developers get “augmented workflow” (faster execution), while non-technical users get “holy crap, I’m a developer workflow” (entirely new capabilities previously impossible).

Improved design-engineering collaboration
Better communication and faster problem-solving because designers understand system constraints and possibilities upfront.

Top tips from the Product Design team

Get proper setup help from engineers
Have engineering teammates help with initial repository setup and permissions - the technical onboarding is challenging for non-developers, but once configured, it becomes transformative for daily workflow.

Use custom memory files to guide Claude’s behavior
Create specific instructions telling Claude you’re a designer with little coding experience who needs detailed explanations and smaller, incremental changes, dramatically improving the quality of Claude’s responses and making it less intimidating.

Leverage image pasting for prototyping
Use Command+V to paste screenshots directly into Claude Code - it excels at reading designs and generating functional code, making it invaluable for turning static mockups into interactive prototypes that engineers can immediately understand and build upon.

Claude Code for RL engineering

The RL Engineering team focuses on efficient sampling in RL and weight transfers across the cluster. They use Claude Code primarily for writing small to medium features, debugging, and understanding complex codebases, with an iterative approach that includes frequent checkpointing and rollbacks.

Main Claude Code use cases

Feature development with supervised autonomy
The team lets Claude Code write most of the code for small to medium features while providing oversight, such as implementing authentication mechanisms for weight transfer components. They work interactively, allowing Claude to take the lead but steering it when it goes off track.

Test generation and code review
After implementing changes themselves, they ask Claude Code to add tests or review their code. This automated testing workflow saves significant time on routine but important quality assurance tasks.

Debugging and error investigation
They use Claude Code to debug errors with mixed results - sometimes it identifies issues immediately and adds relevant tests, while other times it struggles to understand the problem, but overall provides value when it works.

Codebase comprehension and call stack analysis
One of the biggest changes in their workflow is using Claude Code to get quick summaries of relevant components and call stacks, replacing manual code reading or extensive debugging output generation.

Kubernetes operations guidance
They frequently ask Claude Code about Kubernetes operations that would otherwise require extensive Googling, getting immediate answers for configuration and deployment questions.

Claude Code for RL engineering

Development workflow impact

Experimental approach enabled
They now use a “try and rollback” methodology, frequently committing checkpoints so they can test Claude’s autonomous implementation attempts and revert if needed, enabling more experimental.

Documentation acceleration
Claude Code automatically adds helpful comments that save significant time on documentation, though they note it sometimes adds comments in odd places or uses questionable code organization.

Speed-up with limitations
While Claude Code can implement small-to-medium PRs with “relatively little time” from them, they acknowledge it only works on first attempt about one-third of the time, requiring either additional guidance or manual intervention.

Top tips from the RL Engineering team

Customize your Claude.md file for specific patterns
Add instructions to your Claude.md file to prevent Claude from making repeated tool-calling mistakes, such as telling it to “run pytest not run and don’t cd unnecessarily - just use the right path.” This significantly improved consistency.

Use a checkpoint-heavy workflow
Regularly commit your work as Claude makes changes so you can easily roll back when experiments don’t work out. This enables a more experimental approach to development without risk.

Try one-shot first, then collaborate
Give Claude a quick prompt and let it attempt the full implementation first. If it works (about one-third of the time), you’ve saved significant time. If not, then switch to a more collaborative, guided approach.

Claude Code for legal

The Legal team discovered Claude Code's potential through experimentation, and a desire to learn about Anthropic's product offerings. Additionally, one team member had a personal use case related to creating accessibility tools for family and work prototypes that demonstrate the technology's power for non-developers.

Main Claude Code use cases

Custom accessibility solution for family members
Team members have built communication assistants for family members with speaking difficulties due to medical diagnoses. In just one hour, they created a predictive text app using native speech-to-text that suggests responses and speaks them using voice banks, solving gaps in existing accessibility tools recommended by speech therapists.

Legal department workflow automation
They created prototype "phone tree" systems to help team members connect with the right lawyer at Anthropic, demonstrating how legal departments can build custom tools for common tasks without traditional development resources.

Team coordination tools
Managers have built G Suite applications that automate weekly team updates and track legal review status across products, allowing lawyers to quickly flag items needing review through simple button clicks rather than spreadsheet management.

Rapid prototyping for solution validation
They use Claude Code to quickly build functional prototypes they can show to domain experts (like showing accessibility tools to UCSF specialists) to validate ideas and identify existing solutions before investing more time.

Claude Code for legal

Work style and impact

Planning in Claude.ai, building in Claude Code
They use a two-step process where they brainstorm and plan with Claude.ai first, then move to Claude Code for implementation, asking it to slow down and work step-by-step rather than outputting everything at once.

Visual-first approach
They frequently use screenshots to show Claude Code what they want interfaces to look like, then iterate based on visual feedback rather than describing features in text.

Prototype-driven innovation
They emphasize overcoming the fear of sharing "silly" or "toy" prototypes, as these demonstrations inspire others to see possibilities they hadn't considered.

Security and compliance awareness

MCP integration concerns
As product lawyers, they immediately identify security implications of deep MCP integrations, noting how conservative security postures will create barriers as AI tools access more sensitive systems.

Compliance tooling priorities
They advocate for building compliance tools quickly as AI capabilities expand, recognizing the balance between innovation and risk management.

Top tips from the Legal Department

Plan extensively in Claude.ai first
Use Claude's conversational interface to flesh out your entire idea before moving to Claude Code. Then ask Claude to summarize everything into a step-by-step prompt for implementation.

Work incrementally and visually
Ask Claude Code to slow down and implement one step at a time so you can copy-paste without getting overwhelmed. Use screenshots liberally to show what you want interfaces to look like.

Share prototypes despite imperfection
Overcome the urge to hide "toy" projects or unfinished work - sharing prototypes helps others see possibilities and sparks innovation across departments that don't typically interact.