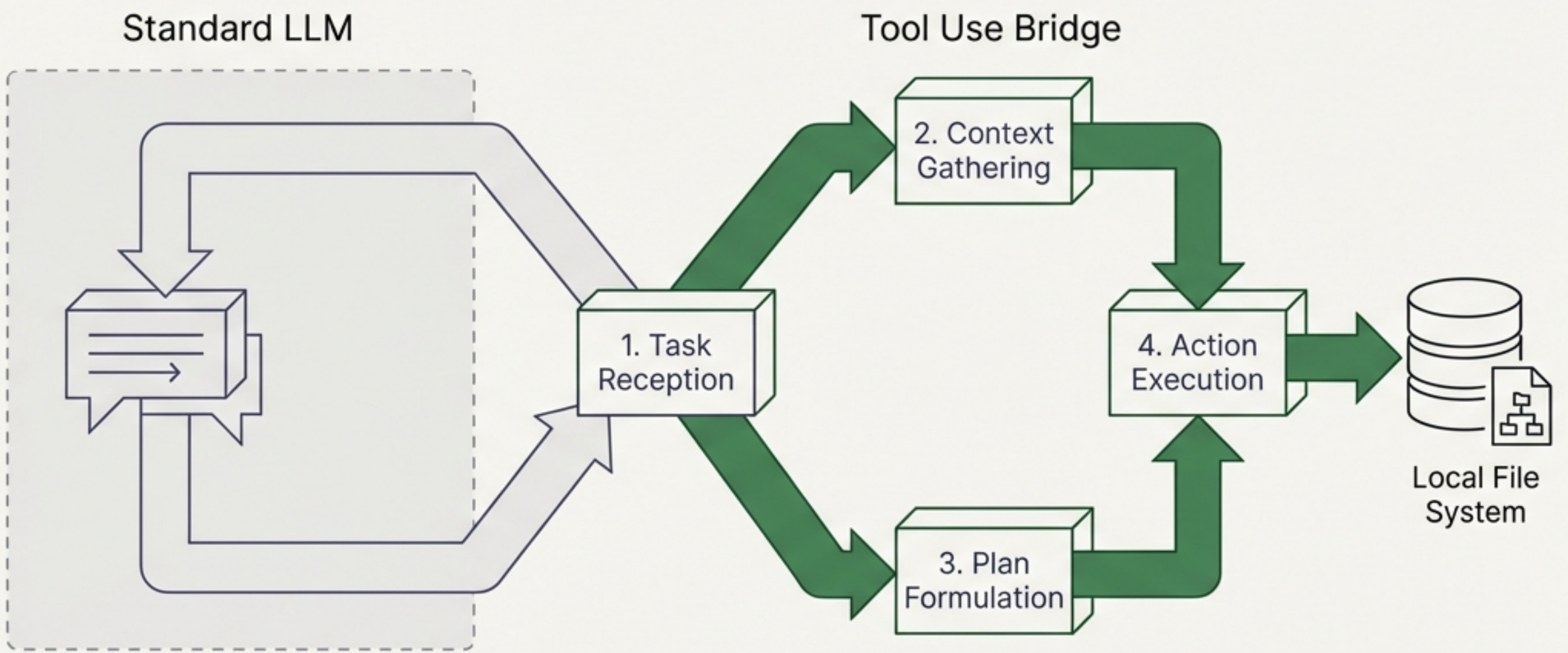
Claude Code: The Extensible AI Assistant

Architecting intelligent, tool-driven development workflows.

Claude Code scales with your team's engineering complexity through programmable tool expansion, not fixed conversational capabilities.

The core engine escapes the text barrier via the Tool Use System



Chalk JS Optimisation: Analysed profiling tools and implemented fixes for a **3.9x** throughput improvement.

Jupyter Data Analysis: Executed Python cells iteratively for CSV churn analysis.

Guardrail: Direct local code search eliminates the security risks of external server indexing.

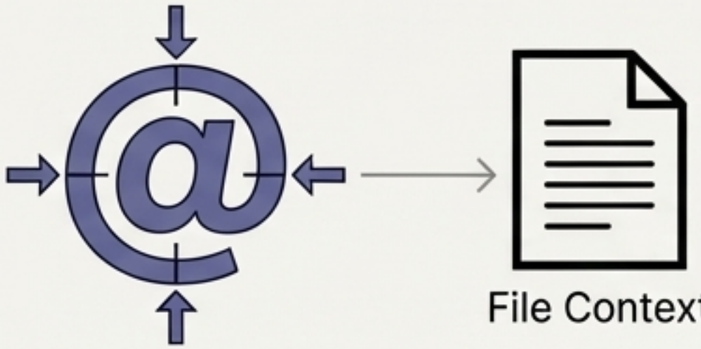
Defining operational boundaries with the Claude.md hierarchy

Project Level		Local Level		Machine Level	
Location	Root directory	**Location**	.Claude directory (Ignored by Git)	**Location**	Global system path
Visibility	Shared with team, committed to source control	**Visibility**	Personal to the individual developer	**Visibility**	User's machine only
Purpose	Architectural summaries, database schemas, and critical project rules	**Purpose**	Individual workflow preferences and local environment quirks	**Purpose**	Global instructions applied universally across all active projects

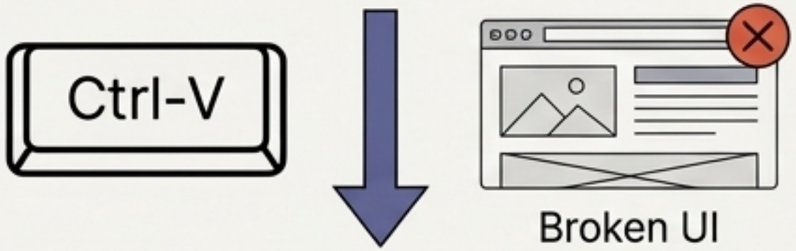
Best Practice: Always reference critical schemas in Project-level Claude.md to guarantee they are included in every request context.

Precision targeting and persistent memory management

Targeted Injection

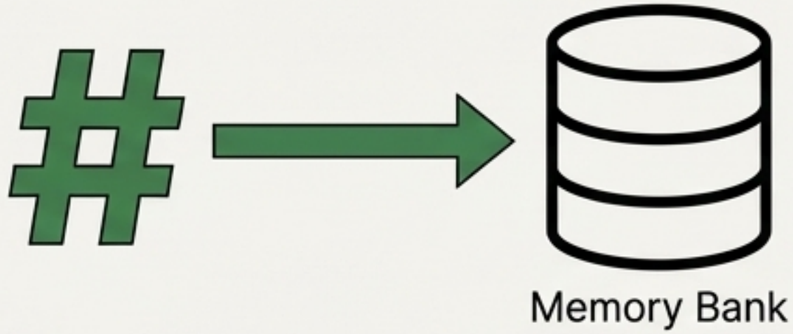


Provide just enough relevant information. Use @ for exact file context, bypassing wide codebase searches.



Use **Ctrl-V** (not Command-V on macOS) to drop visual UI screenshots directly into the terminal context.

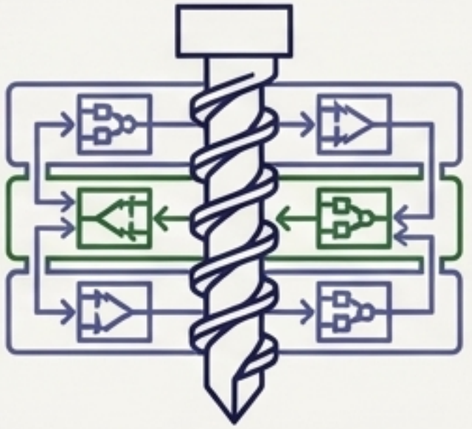
Persistent Learning

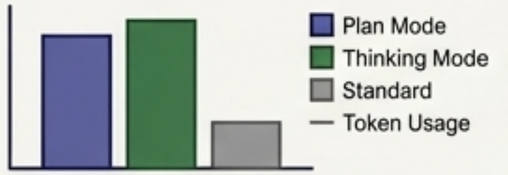


Use # to write rules directly into `Claude.md` using natural language.

Prevents repeated errors in future sessions by continuously updating the boundary context file.

Allocating cognitive **budget:** Plan Mode vs. Thinking Mode

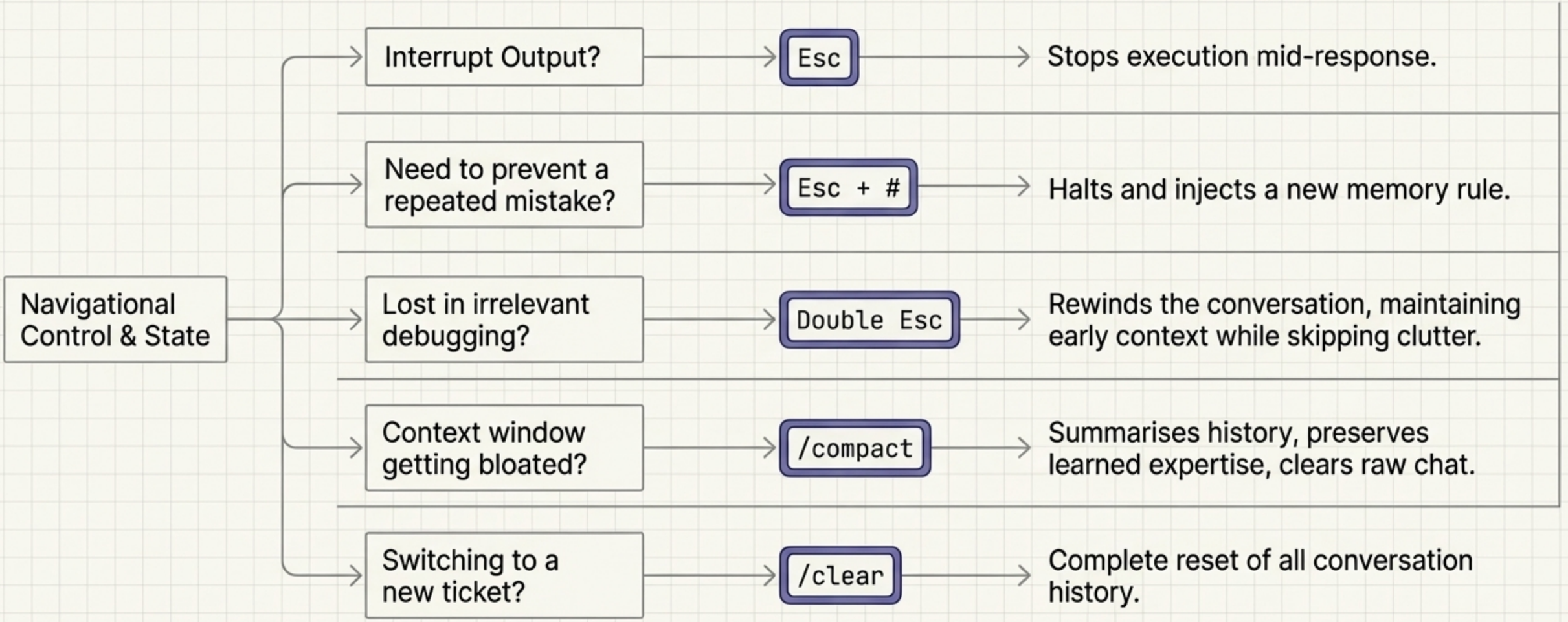
Plan Mode (Horizontal / Breadth)	Thinking Mode (Vertical / Depth)
	
Trigger: `Shift + Tab x2`	Trigger: `'Ultra think'`
- Used for horizontal tasks. - Multi-step features requiring wide codebase understanding. - Generates detailed implementation plans before execution.	- Used for vertical tasks. - Tackles tricky logic and advanced debugging. - Provides extended reasoning budget for specific constraints.



Resource Note: Both modes seamlessly integrate with Git to stage and commit changes, but consume additional tokens. Deliberate usage is required based on task complexity.

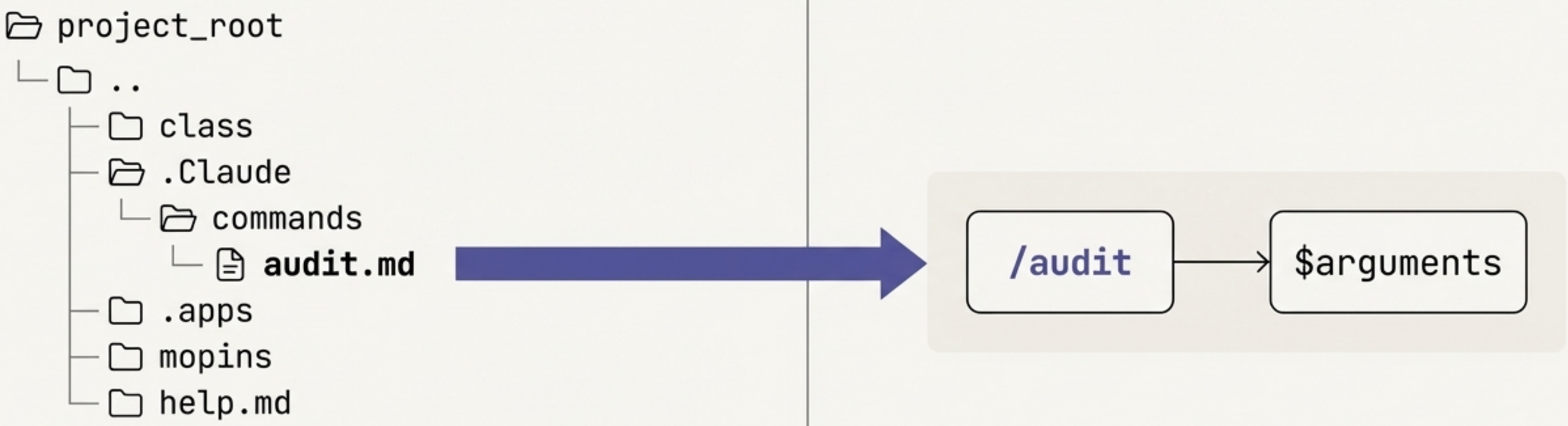
Editorial Engineering Whitepaper

Navigational control and conversation state management



Takeaway: Maintain focus, reduce distracting context, and preserve relevant engineering knowledge without restarting the underlying engine.

Automating team workflows with custom execution commands



Architecture

Markdown files placed in the `.Claude/commands/` folder become native executable commands. Requires application restart to activate.

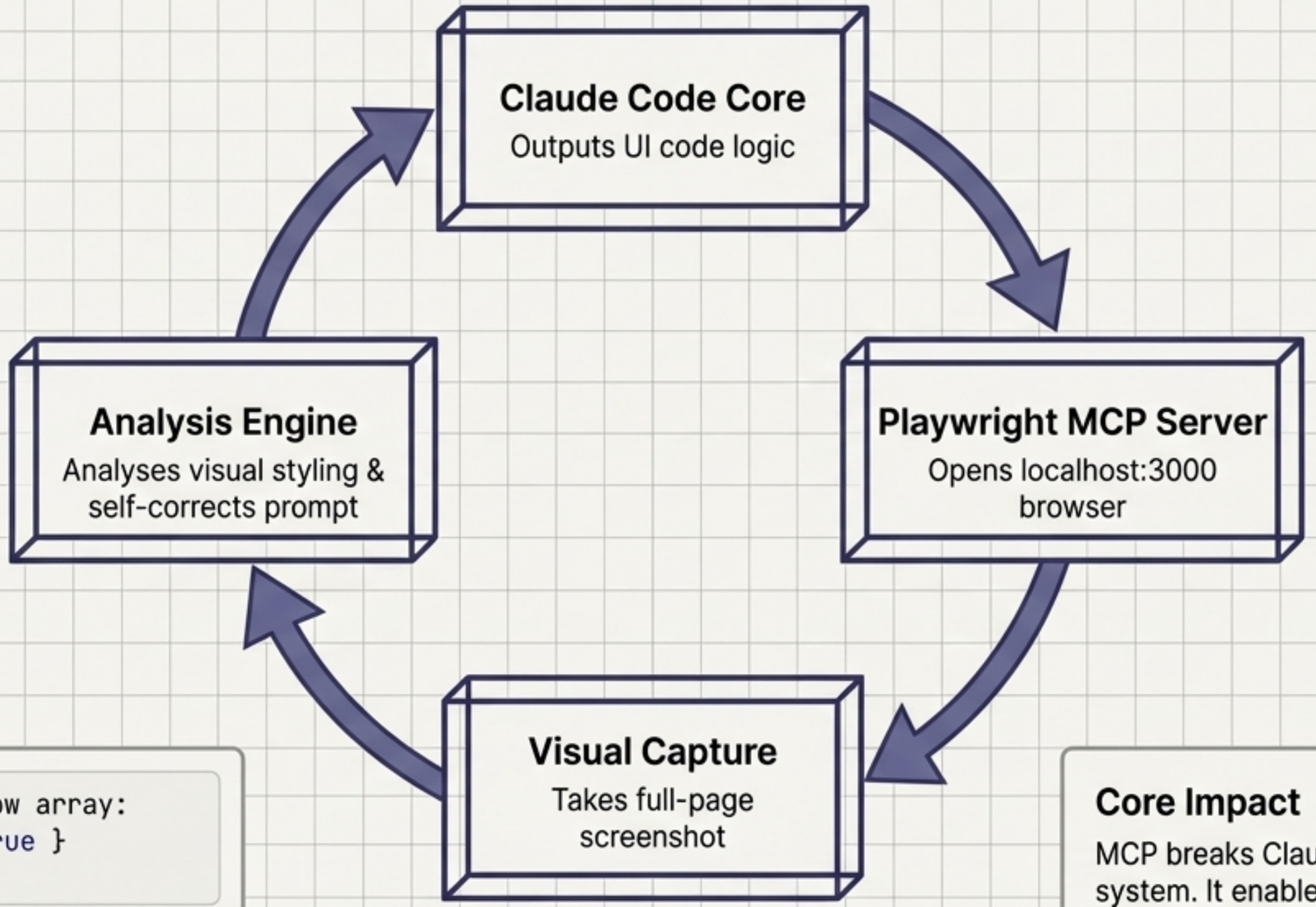
Execution

Combine the command name with a string input block for runtime parameters (e.g., `/audit src/components`).

Use Cases

Standardising dependency auditing, test generation, and automated vulnerability fixes across the engineering team.

Extending operational reach via the Model Context Protocol (MCP)




```
settings.local.json allow array:
  {'MCP__playwright': true }
```

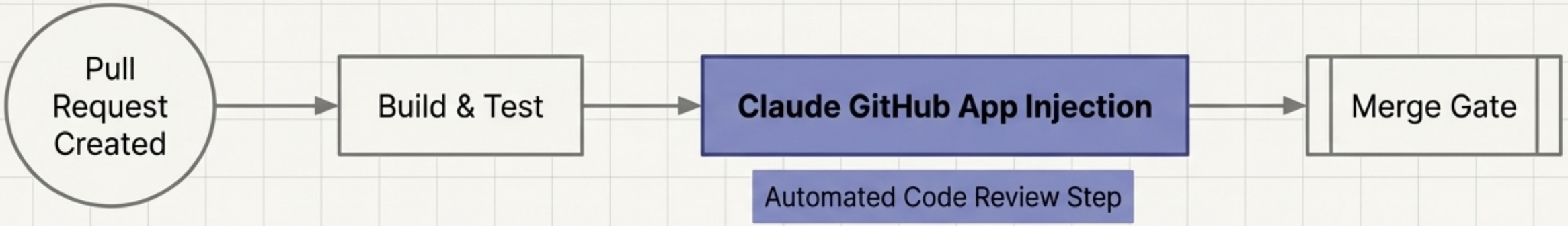
Bypasses manual approval for continuous execution.

Core Impact

MCP breaks Claude out of the static file system. It enables complex, multi-step operations involving live external environments and automated visual QA loops.

Editorial Engineering Whitepaper

Asynchronous team integration within GitHub Workflows

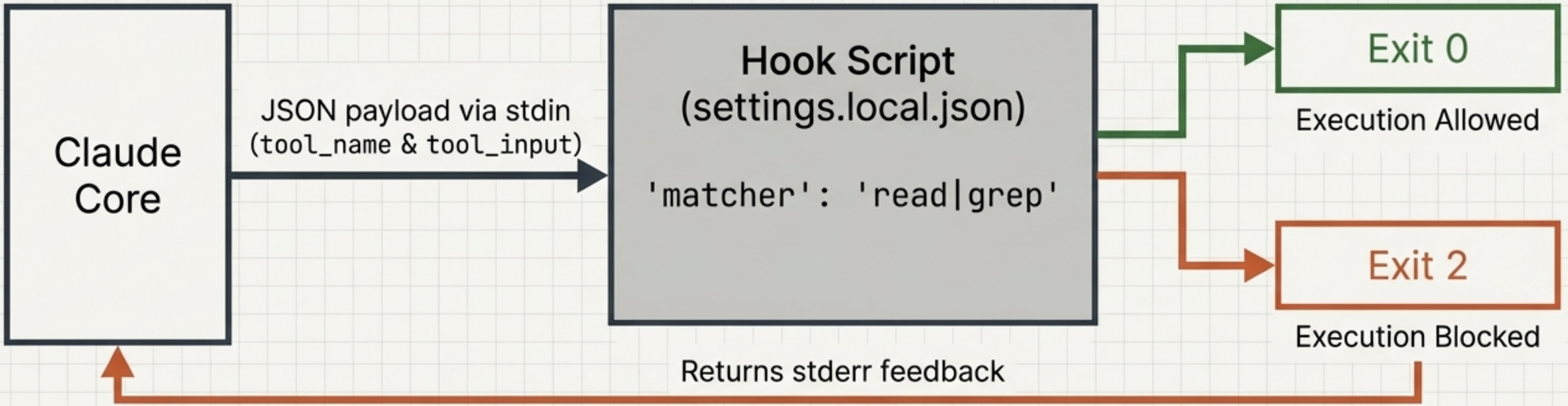


Feature Breakdowns

1. **Mention Support**
   - "Use **@Claude** in issues and PRs to assign direct debugging tasks to the agent." (JetBrains Mono for @Claude)
2. **Security Analysis**
   - "Automated infrastructure review (e.g., detecting PII exposure risks in Terraform AWS Lambda outputs)." (JetBrains Mono for PII exposure risks in Terraform AWS Lambda outputs)
3. **Custom Workflows**
   - "Configurable via **.github/workflows**, allowing cloud-based MCP integrations for UI testing." (JetBrains Mono for .github/workflows)

Explicit Permission Requirement: All tools must be explicitly listed in the actions configuration. There is no automatic tool inheritance in the cloud environment.

The Hooks Architecture: Intercepting and evaluating tool calls



Pre-tool Hooks

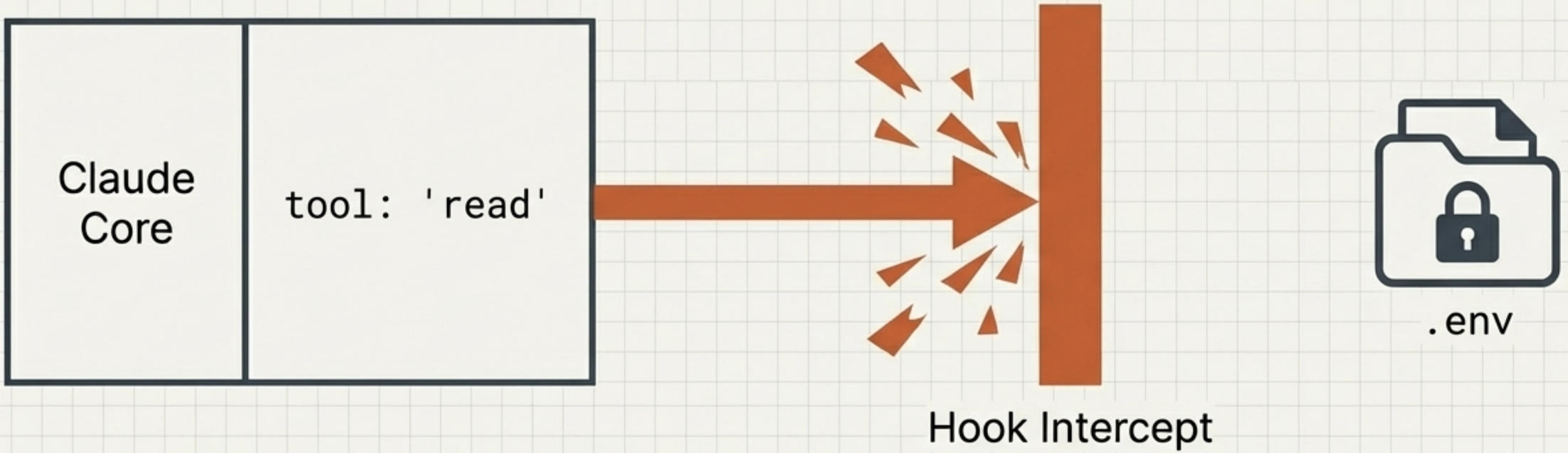
Act as blocking security gateways before any action is taken.

Post-tool Hooks

Act as follow-up operations and continuous feedback loops after execution.

Editorial Engineering Whitepaper

Security Guardrails: Implementing a pre-hook file blocker



1. Monitor

Target the `read|grep` tools in the configuration matcher.

2. Inspect

Script parses the JSON payload and checks if `tool_input.path` includes `'.env'`.

3. Block

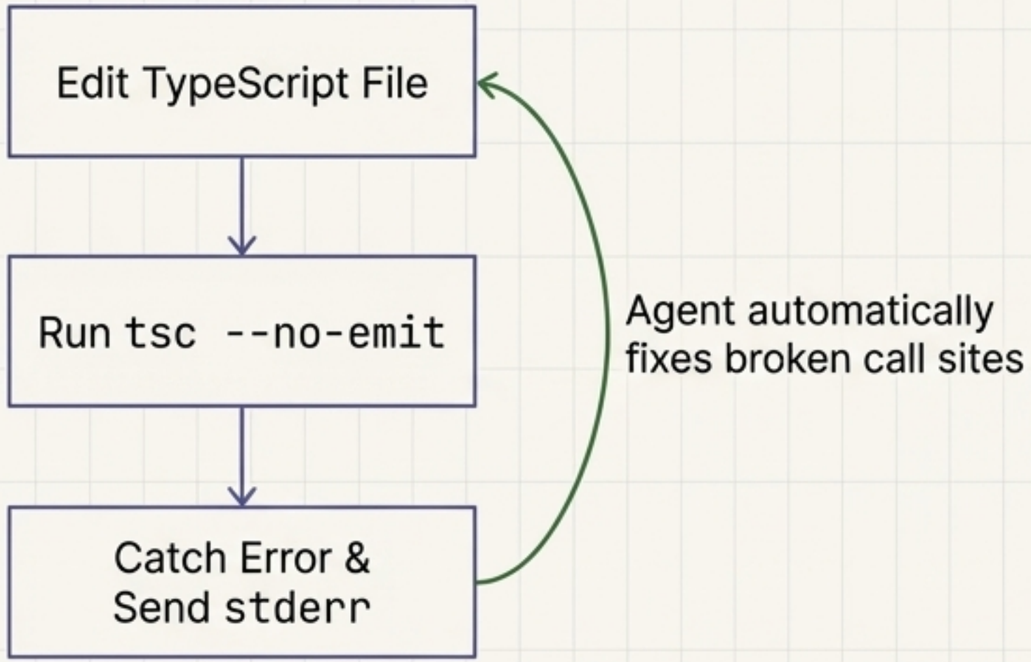
If true, instantly trigger process `exit(2)`.

4. Feedback

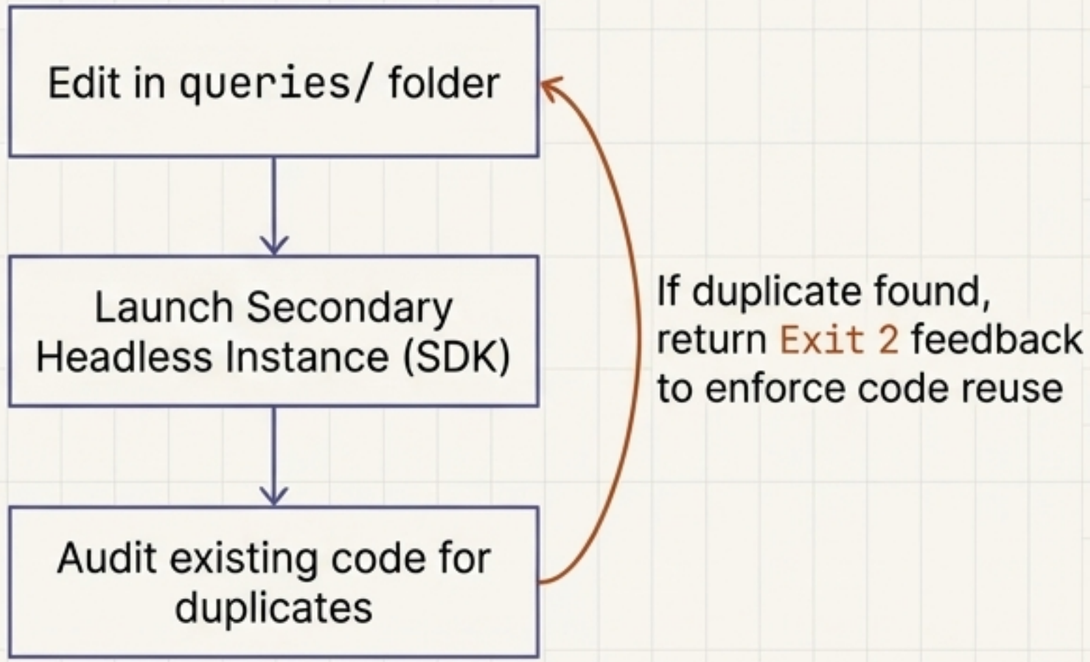
Send `console.error()` message via `stderr` so the agent understands why the action was rejected.

Automated self-correction using post-hook feedback loops

1. TypeScript Type Checker

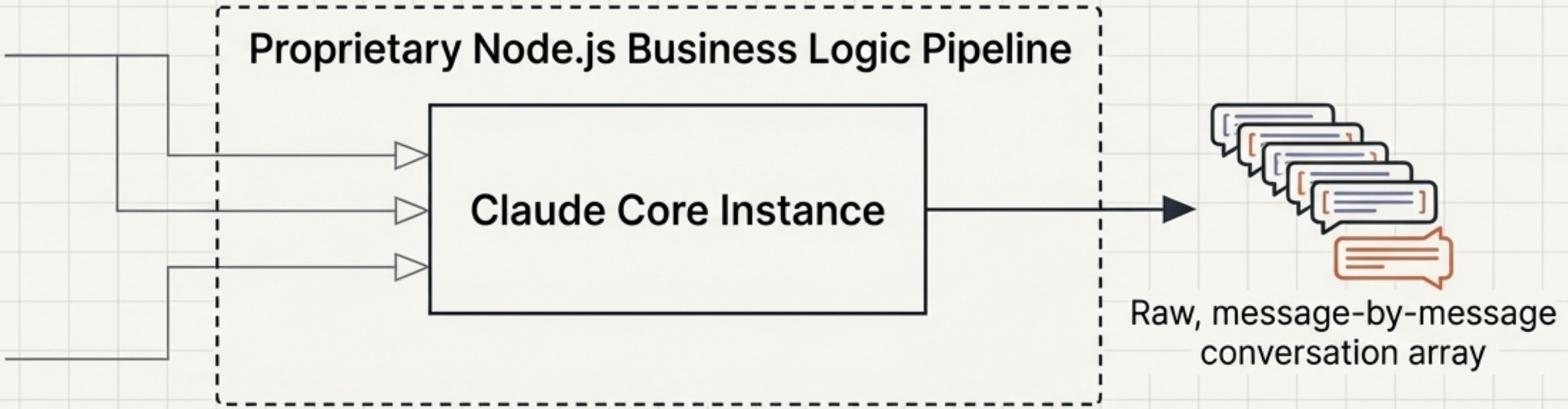


2. Duplicate Query Prevention



Architectural constraint: Secondary audits drastically improve codebase cleanliness but introduce additional token cost and execution latency. **Restrict to critical directories only**.

Embedding intelligence with the TypeScript SDK



Primary Use Case

Adding intelligence as a modular component to existing helper commands, custom scripts, and hooks, rather than standalone usage.

Permission Architecture

The SDK is strictly `read-only` by default (`files`, `directories`, `grep` `operations`) to ensure pipeline safety.

Implementation Pattern

Write capabilities must be explicitly injected via the `options.allowTools` array (e.g., manually adding the 'edit' tool during the `query` call).

Synthesis: Orchestrating the Claude Code Ecosystem

Context Layer

`Claude.md` boundaries, `@file` targets, `# memory` rules.

Cognitive Control

`Plan` vs. `Think` modes, Escape state management

Tool Use Engine loop

Developer

Guardrails

Pre/Post Hooks, exit codes, strict SDK permissions.

Extensibility

MCP Servers, Custom Commands, GitHub Actions.

Claude Code is not a terminal wrapper for an LLM; it is a programmable, modular middleware layer that translates natural language intent into secure, boundaried system operations.

Strategic directives for engineering integration

1

Context is Configuration

Effective autonomy relies on precise boundaries. Standardise `Claude.md` files across repositories and enforce `@` targeting to minimise context bloat.

2

Programmable Guardrails

Transform unpredictable behaviours into safe, iterative loops. Utilise the Hook architecture (`exit 2`, `stderr`) enforce security boundaries (`.env`) and CI standards (`tsc --no-emit`).

3

Limitless Extensibility

The assistant scales with your infrastructure. From local `Playwright` browser automation to GitHub PR Terraform reviews, expand capability by adding tools, not just modifying prompts.