Guardrails, Not Guesswork

Shipping Drupal Features with Claude Code

Drupal Asheville Camp · Ryan K Olsen
ryankolsen@gmail.com, linkedin.com/in/ryan-olsen-a5463a18

Ryan Olsen

Software Engineer · Portland Webworks

🛠️ The Journey
React/Redux → Java → Drupal
~4 years at Portland Webworks

👨‍👧 Off the Clock
Husband & girl dad
Audiobooks & Dungeon Crawler Carl

💼 Origin Story
Escaped from sales
No regrets whatsoever

⛺ Community
Drupal Asheville Camp
2025 attendee → 2026 speaker

Why Guardrails?

The real risks aren't just dramatic failures. They're the quiet, everyday ones:

- Editing a file not tracked in config — disappears on the next import
- Logic in Twig instead of a preprocess hook — AI didn't know your conventions
- Context window grows unmanaged — quality degrades before you notice
- Contrib module files edited directly instead of patched
- Security implications missed when env context wasn't provided

Guardrails = code quality + token efficiency + consistency + confidence

LLMs Are Stateless

Every new conversation is a blank slate.

Claude Code is like a new developer joining your team every 20 minutes with zero institutional knowledge.

It doesn't remember:

- What you built last session
- Your team's naming conventions
- That you use SDCs, not block templates
- The contrib patches you've applied

Fix: CLAUDE.md + skills give it institutional knowledge automatically.

The Knowledge Cutoff

The model's Drupal knowledge is frozen.

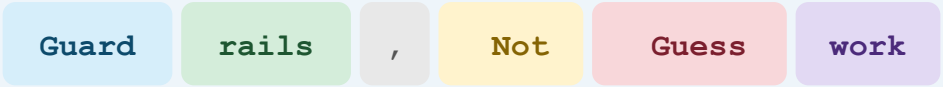
- It may not know the module you're using
- APIs that changed after its training cutoff
- Your site's specific architecture
- The contrib patches you've applied

You can't trust it to have the most up to date knowledge

Resource: aicodingdictionary.com

What Is a Token?

The unit of text a language model reads and writes — not a word, not a character.



= 6 tokens (for "Guardrails, Not Guesswork")

- ~750 words ≈ 1,000 tokens
- Code is token-hungry — indentation, variable names, syntax all count
- Your CLAUDE.md and skills are injected as tokens on every single request
- Files you @-mention are read in full — every character is a token

How you're billed

INPUT tokens

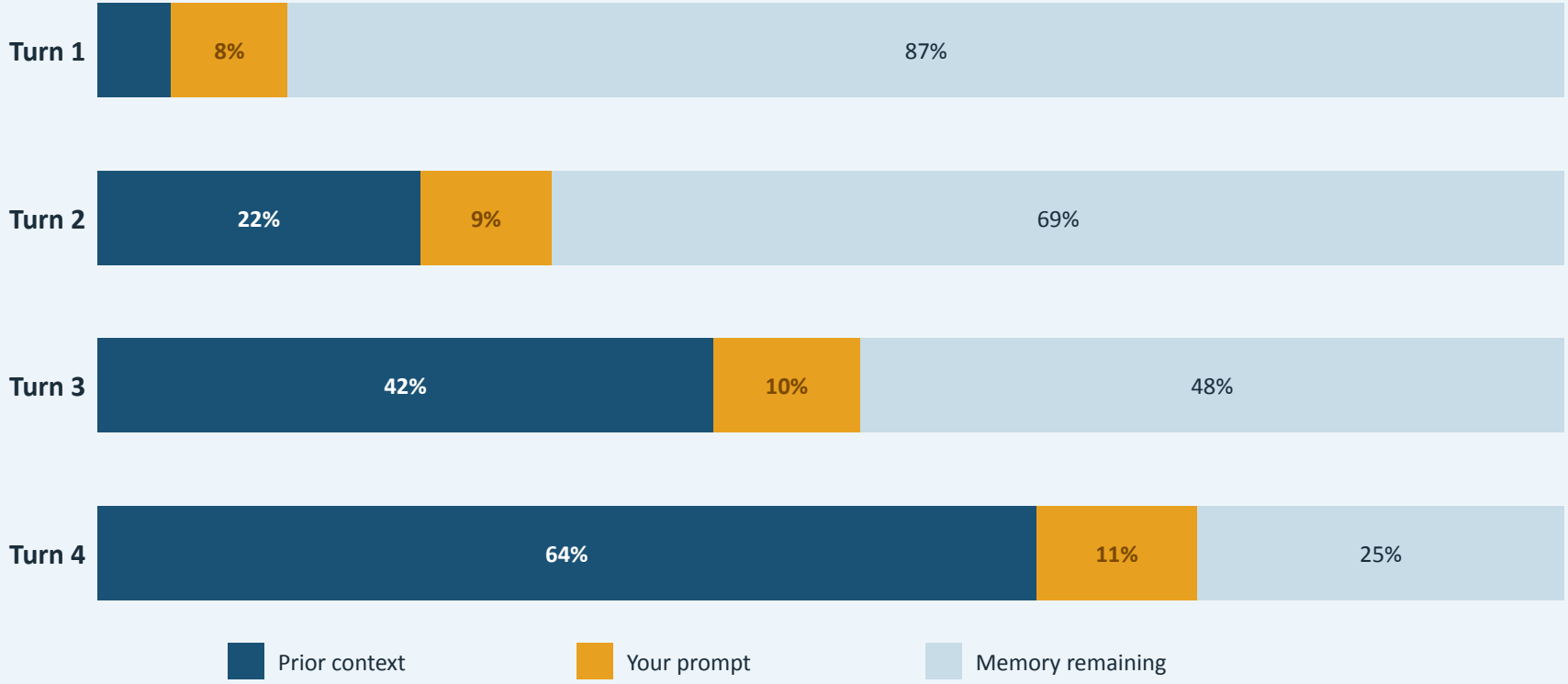
Your message + entire conversation history
+ attached files + CLAUDE.md + skills

OUTPUT tokens

Claude's response
(costs 3–5× more per token than input)

Every conversation turn re-sends your entire history as input tokens — context grows every turn, and so does the cost. This is why context window management isn't just a performance tip: it directly affects quality and your bill.

Context Grows Fast — Stay in the Smart Zone ~150K



Tip: watch your token count in the status bar — act before you hit the dumb zone, not after you notice quality dropping

~/.claude/statusline.sh — Monitor Your Context Live

```
[claude-sonnet-4-5] Chat: 87K | ↑ 72K ↓ 15K | $0.43 | 2:30 PM: 34% | Week: 12%
```

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
USED=$(echo  "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
COST=$(echo  "$input" | jq -r '.cost.total_cost_usd // 0')
IN_TOK=$(echo "$input" | jq -r '.context_window.total_input_tokens // 0')
OUT_TOK=$(echo "$input" | jq -r '.context_window.total_output_tokens // 0')

# Detect /clear — reset baseline when context drops to 0
if [ "$USED" = "0" ] && [ "$LAST_USED" -gt 0 ]; then
    BASELINE_IN=$IN_TOK; BASELINE_OUT=$OUT_TOK
fi

CHAT_IN=$((IN_TOK - BASELINE_IN))
CHAT_OUT=$((OUT_TOK - BASELINE_OUT))
CHAT_TOTAL=$((CHAT_IN + CHAT_OUT))

printf "[%s] Chat: %s | ↑ %s ↓ %s | $%.2f%s" \
    "$MODEL" "$(humanize $CHAT_TOTAL)" "$(humanize $CHAT_IN)" "$(humanize $CHAT_OUT)" "$COST" "$LIMIT_PART"
```

Full script: ~/.claude/statusline.sh | https://github.com/Ryankolsen/drupal-ai-demo/blob/main/.claude/statusline.sh

Engineering Matters More Than Ever

AI doesn't replace your engineering judgment — it amplifies it. Good or bad.

Principles to enforce — and encode in your skills

- **Clean code & DRY**

Logic in preprocess hooks and services — never in Twig. If the AI generates it in a template, your skill should catch it.

- **Established Drupal patterns**

Plugin system, service container, dependency injection, Queue API. Guide the AI toward the pattern, not the shortcut.

- **Test-first (red → green)**

Write the PHPUnit or Kernel test before the implementation. The AI can write both — make it write the test first.

- **Vertical slices as tracer bullets**

Route → controller → service → SDC → test. Get one thin slice working end-to-end, then expand. AI stays on track when scope is tight.

The AI mirror effect

Well-architected codebase

Services, SDCs, PHPStan compliance, hooks in the right place, consistent naming → AI produces code that matches the pattern

Tangled codebase

Logic in Twig, inconsistent patterns, no tests, spaghetti hooks → AI reproduces and compounds every bad habit it finds

Tip: enforce PHPStan + PHPCS via pre-commit hooks — Claude Code will run them automatically and self-correct

Your codebase is the AI's reference for your project. It will reproduce what it sees — good architecture and bad habits alike.

The Implication

Your expertise has to live somewhere the AI can find it.

CLAUDE.md

Permanent project briefing: architecture, conventions, what not to touch

Skills

Reusable instruction sets encoding your team's standards — applied on every task

settings.json

Hard rules: which commands the AI can and can't run in your project

CLAUDE.md, Rules & Settings

Unlike skills, these run automatically on every session — no slash command, no invocation needed.

`CLAUDE.md`

What Claude always knows

- Loaded at the start of every session automatically
- Keep as slim as possible
- Patterns to follow and patterns to avoid
- Avoid adding project architecture, module list, this can be inferred
- Correct repeated mistakes that claude makes

Example rule: "We use SDCs, not block templates. Logic goes in preprocess hooks, not Twig."

`settings.json`

What Claude is allowed to do

- Lives at .claude/settings.json — committed to git
- Allow or deny specific bash commands by pattern
- Prevent editing contrib modules or untracked files
- Set which tools Claude can and can't run
- Apply to the whole team — everyone shares the same rules

*Example: deny: rm -rf, deny: edit web/modules/contrib/**

`settings.local.json`

Your personal overrides

- Lives at .claude/settings.local.json — gitignored
- Personal overrides that don't affect the team
- Great for allowing tools you use that others don't
- Also: ~/.claude/settings.json for global user defaults
- Hierarchy: user → project → local (most specific wins)

Example: Allow docker commands locally while the team's settings.json restricts them

Skills = how to do specific things. CLAUDE.md = what Claude always knows. settings.json = what Claude is allowed to touch. Three layers, one guardrail system.

What Is a Skill?

If you've typed the same prompt more than once — that's a skill waiting to be written.

- **A skill is a .md file**

Lives in .claude/skills/ — plain text, versioned like any code

- **Encode your expertise**

Your conventions, your stack, your rules — applied every time, not just when you remember to ask

- **Claude can write them for you**

"Turn this prompt into a reusable skill" — then refine it in the next chat

- **Claude can edit its own skills**

Show Claude a skill and ask it to improve, tighten, or add a rule — skills get better over time

Concept credit: Matt Pocock — github.com/mattpocock/skills

How to invoke

Manual — you call it:

`/write-a-prd`

Auto — Claude decides:

Claude reads the skill's description field and invokes it automatically when the task matches — no slash command needed

```
---
name: do-work
description: Main work skill. Apply to
  any implementation task.
---
Never put logic in Twig templates.
```

Skills encode your workflow expertise — the how-to for specific tasks. They work alongside CLAUDE.md (what Claude always knows) and settings.json (what Claude is allowed to do). Together, these three are your complete guardrail system.

Skills in My Daily Workflow

Repetitive tasks become slash commands. Here's what I invoke most on Drupal projects.

PLAN	BUILD	SHIP
`/write-a-prd` Define the feature as a Product Requirements Doc before any code is written	**`/do-work`** Implement a ticket end-to-end — reads context, writes code, runs checks	**`/drupal-code-review`** Scan for side effects, bad patterns, duplication, and Drupal anti-patterns
`/prd-to-plan` Convert a PRD into a structured implementation plan with phases and tasks	**`/commit`** Draft a Conventional Commit message from staged changes	**`/write-a-pr-review`** Generate a PR description and summary from the diff
`/prd-to-issues` Explode a PRD into individual tickets ready to drop into your backlog	**`/handoff`** Document state, decisions, and next steps for the next developer (or future you)	

Each skill is a .md file in .claude/skills/ — version-controlled, shareable across the team, and improvable over time.

Helpful Claude Code Commands

Navigation & Session

`/resume` — pick up a previous session

`/clear` — fresh context, same project

`/memory` — review what Claude remembers

`/rename` — rename the current session

Context & Skills

`@filename` — pull a file into context

`/skills` — list available skills

`/handoff` — hand off to a fresh Claude — better than /compact

`/ide` — connect to VS Code / Cursor

Keyboard Shortcuts

`Option+Enter` — newline in input (after /terminal-setup)

`ctrl+S` — stash your current input

`ctrl+V` — paste an image | cmd+V pastes text

`esc esc` — restore to a previous point in the conversation

Bash & Terminal

`/terminal-setup` — enables Option+Enter for multi-line input

`/usage` — check token & usage stats

`!bash` — shortcut to run a bash command

`ctrl+Z` — suspend chat to run a command yourself

Tips & Tricks

Rename every chat

Use /rename — ticket # + short description

Wispr Flow — voice prompting

Dictate prompts instead of typing — wisprflow.ai

Writing & trimming skills

/write-a-skill and /trim-a-skill — keep skills under ~100 lines

Claude Desktop for non-code work

Add your notes as project context — docs, slides, specs, proposals

Always run cex after cim

After AI updates config and you run cim, always follow with cex

Give the AI your branch context

Ask the AI to review all changes on your branch before the next task

Resources

AI Coding Dictionary *aicodingdictionary.com*

Matt Pocock's Skills (workflow inspiration) *github.com/mattpocock/skills*

7 Bad Twig Practices *menetray.com/en/blog/7-bad-twig-practices-are-breaking-your-drupal-and-how-fix-them*

Wispr Flow (voice prompting) *wisprflow.ai*

Browser MCP *browsermcp.io*

3Blue1Brown: Neural Networks (how LLMs actually work) *youtu.be/aircAruvnKk*

Demo repo *github.com/Ryankolsen/drupal-ai-demo*

Skills plugin repo *github.com/Ryankolsen/drupal-guardrails-skills*
Open Claude and run: *'/plugin marketplace add'*
when prompted: 'ryankolsen/drupal-guardrails-skills'