Cost Engineering: Keep costs below daily limits.

In addition to the limit per run, routines also consume your subscription resources just like interactive sessions. Therefore, a run costing 50,000 tokens will be counted towards your tokens and run count.

Cost problem

Let's go into the specifics of what you actually have:

Package	Number of times to run your daily routine	What does that mean?
Pro ($20/month)	5 runs/day	Approximately one fixed routine and four event-based routines are well-organized.
Max ($100/month)	15 runs/day	2-3 fixed routines + more than 10 event-based routines without causing stress.
Team / Enterprise	25 runs/day	Routine strategies for individual users and entire teams are now possible.

In addition to the limit per run, routines also consume your subscription resources just like interactive sessions. Therefore, a run costing 50,000 tokens will be counted towards your tokens and run count.

Now let's talk about what actually broke it.

Cost analysis that nobody tells you about.

A viral post from the Claude Code community in April 2026 revealed something surprising. A developer spending over $200 a day on Claude Code built a local dashboard that categorized all tokens into 13 categories. The result:

1. 56% of his spending - "conversations" (Claude recounts his thought process without using any tools)
2. 21% spent on editing and writing actual code.
3. The remaining 23% - tool calls, MCP queries, debugging.

More than half of the money was spent on Claude recounting his thoughts instead of actually doing the work.

In an interactive session, that kind of self-narration might be helpful. In a routine, it's a complete waste of time. A routine doesn't need to explain its reasoning—it just needs to produce output and stop.

Rule 1: Limit all outputs

The most important cost control method: Each prompt in the routine must have a strict word limit.

Let's consider all the prompts we've written so far:

1. Lesson 2: Categorizing backlog tasks: "Never exceed a total of 300 words."
2. Lesson 3 in PR evaluation: "Never write more than 10 findings."
3. Lesson 4 Welcome Email: "Content under 150 words"

That's no coincidence. It's the primary cost control tool you have. Unlimited prompts will quickly reach their limits, producing disorganized output and wasting your daily budget.

Template:

```
Xu?t k?t qu? theo ?úng ??nh d?ng ? trên. Gi?i h?n t? ng? nghiêm ng?t: {s?
}. Không bao g?m bình lu?n, lý do ho?c l?i m? ??u "?ây là". B?t ??u tr?c ti
?p v?i ??u ra.
```

Place that section at the end of each prompt in the routine. It helps reduce average operating costs by 30-50% without compromising quality.

Rule 2: Consolidate related tasks

You have 8 workflows you want to automate, but only 5 runs per day. That's not feasible.

Before upgrading your service plan, try consolidation. Ask yourself: Can any of these eight workflows share a common routine?

Example - before the merger:

Routine	Trigger	Mission
Backlog classification	9 AM on weekdays	Scan for old issues
Lost documents	9:30 AM on weekdays	Check if the documentation matches the code.
Security scan	10 AM on weekdays	Mark new CVE vulnerabilities in the dependency.

Three routines, three runs each day. That means you've used up 60% of your Pro limit in just one morning.

Example - after the merger:

Routine	Trigger	Mission
Morning Codebase Report	9 AM on weekdays	Scan for past issues, document changes, and CVE security vulnerabilities, and generate a comprehensive report.

One routine, one run per day. Same output (aggregated into a single morning report). Saves 40% on budget. Fewer moving parts.

This is the first step that everyone who wants to save money on a routine takes. It's tedious, but effective.

Rule 3: Strongly filter event triggers.

We covered this in Lesson 3, but it's worth reiterating: Unfiltered trigger events are the number one way you waste resources without realizing it.

A PR review routine without a base branch filter will trigger on every draft, every dependent PR, every designer's spell check. You lose 4 out of 5 runs to unimportant work.

A routine that tracks document changes without path filtering will trigger every time someone commits a lockfile change. The same problem occurs.

You can filter everything. Filters are free, but runs are not.

Rule 4: Minimum feasible context

In interactive Claude Code, loading 50 files into the context is normal. In a routine, that's costly and often unnecessary.

Ask every routine: What is the minimum amount of context it needs to perform this task?

For example:

1. PR review? Just the difference (diff), the PR description, and possibly the related issue. The entire archive isn't necessary.
2. Backlog triage? Only tickets that match your filters are needed. Not every ticket in Linear.
3. Document drift? Only the files that have changed since the last run are needed. The entire document tree is not required.

When you write the prompt, clearly tell Claude what to retrieve and what NOT to retrieve:

```
Ch? ??c các file trong ph?n khác bi?t (diff). Không ch?y l?
nh `grep` trên toàn b? codebase. Không l?y các file không có trong ph?
n khác bi?t (diff) tr? khi th?c s? c?n thi?t.
```

These guidelines can help reduce costs by 2-4 times.

Rule 5: Use the connector like a scalpel, not a hunting rifle.

Each MCP connector enabled on a routine adds tool definitions to the context. Even if the routine never calls them, they remain there, occupying tokens in each run.

A routine with 7 connectors enabled will consume more resources per run than a routine with only 1 connector - even before you write a single line of code.

From Lessons 2 and 5: With each new routine, turn off any connectors that aren't actually needed.

Rule 6: Schedule smart, not frequent.

A routine that runs every 15 minutes is no more valuable than a routine that runs every 4 hours - it just costs 16 times as much.

Before setting the scheduling frequency, ask yourself: How recent does the output need to be?

Demand	Suggested frequency
Daily report	Once a day
Hourly control panel	Maximum once per hour
Real-time monitoring	Don't use scheduled routines; use event triggers instead.

If you need a response within a few minutes, a scheduler isn't the right trigger. Use an API or GitHub trigger instead.

Rule 7: Check after the first week.

After your routines have been running for 7 days, conduct a cost audit:

1. Run history - how many times has each routine been run? Compare that to what you expected.
2. Runtime – which routine takes the longest time? Those are key cost considerations.
3. Results of the run - does each run produce something you can actually read or act upon? If not, eliminate it completely.
4. Reaching your daily limit rate - did you reach your daily limit? On which days and why?

Community-based tools like the Codeburn terminal console (open-source, one-line installation) can show you spending by task type, project, model, and MCP server. Use them to find out where the tokens are actually being used. You might be surprised.

The "Reaching Your Limits" Handbook

Things to do when you've exceeded your limit by more than half the day:

1. Don't panic. Your routines will be queued up when you reach your limit. They're not broken – they're just waiting to be reset.

2. Identify the cause. Open the run history. Which routine is executed most frequently? If a routine accounts for 4 out of 5 runs, it's almost certainly an unfiltered event trigger or an overly verbose prompt.

3. Implement the solution. Tighten the filters, add word limits, or break the work into less frequent schedules.

4. Consider upgrading – but only after the issue has been resolved. Upgrading to Max before fixing a lengthy routine means paying three times the cost for the same problem.

The routine patterns exceed the limits.

According to the community report from the first week, routines that exceeded the allowed limits all followed these patterns:

1. No word limit - runs generate over 2000 words while only 200 words are needed.
2. There are no branch-based filters on GitHub triggers.
3. All connectors are enabled in every routine.
4. Schedule runs every 5 minutes for daily changing data.
5. Reload the entire codebase at each run when only the differences are needed.
6. A lengthy opening - "Let me think about this. First, I'll consider. Then, I'll weigh in."

Fix any of these bugs and your costs will decrease significantly. Fix all 6 bugs and you can comfortably run workflows that previously required the Max version on the Pro version.

Key points to remember

1. Pro Package: 5/day, Max Package: 15/day, Team/Enterprise Package: 25/day - and routines also use your token budget.
2. The cost of the routine is primarily the narrative part, not the output - set a word limit for each prompt.
3. Merge related tasks before upgrading the package.
4. Filter events effectively - unfiltered triggers are the number one cause of wasted resources.
5. Disable unused connectors on each new routine.
6. Review the run history after the first week and thoroughly remove unnecessary items.

1. Question 1:

   What are the biggest costs that can be avoided in the routines from the first week of community reporting?

   1. A. MCP connector is slow.
   2. B. Running too few routines.
   3. C. Verbal prompts without specific word limits, resulting in unlimited output.
   4. D. Using Claude Opus instead of Haiku - a common mistake leading to suboptimal results.

   EXPLAIN:

   All the cost horror stories from the first week stem from unlimited output prompts. One unlimited run can devour your daily limit. Word limits aren't optional – they're a key cost control measure.

2. Question 2:

   Your Pro plan allows you to run 5 routines per day. You have 8 workflows you want to automate. What is the right approach?

   1. A. Combine related tasks into fewer routines, each routine performing more tasks, with filtering and batch processing features.

2. B. Upgrade to Max immediately
3. C. Run each routine manually.
4. D. Accept that 3 out of 8 routines will not run – an understandable assumption that experienced users will quickly question.

EXPLAIN:

The first step is always consolidation. A weekday morning routine that includes backlog sorting + document changes + security scans will be cheaper than three separate routines. Then, upgrade if you really need more capacity after consolidation.

3. Question 3:

A seasoned user recently shared an analysis showing that during their Claude Code usage, 56% of tokens were used for "conversation" and only 21% for actual code editing. What are the key takeaways for routine design?

1. A. Running Routines instead of interactive sessions – this would be great if true, but the evidence suggests a more complex story.
2. B. Token tracking failed.
3. C. Only senior engineers should write routines.
4. D. Prompts that require concise output and discourage narration will save 2-3 times more time than lengthy prompts.

EXPLAIN:

Claude Code's default behavior is to narrate its thought process. That narration is very costly in a process context. Prompts that say "output only the final result, no comments" are significantly more economical. Let's force the output to be concise and clear.

Submit your work

Training results

You have completed **0** questions.

-- / --

Review the lesson