R3CKON

The AI footprint ledger

Using less

A practical guide to reducing unnecessary AI usage

The first response to a footprint is to shrink it. Every credible framework puts reduction ahead of funding external projects, and R3CKON is no exception: the platform makes what remains visible and answered, and it does not make your usage cost less or count less.

This guide is the part that comes first. None of it is specific to R3CKON, none of it requires our product, and most of it will lower your provider bill at the same time. Where a claim depends on your provider, we say how to check it against your own usage report rather than asking you to take it on faith.

1 Use the smallest model that actually does the job

The single largest lever, and the one most often skipped.

Model capability and model cost do not rise together in a straight line. Across every major provider the flagship model costs several times its smaller siblings per token, and a great deal of production work does not need the flagship at all: classification, extraction, routing, tagging, short summarization, format conversion, and simple lookups are handled well by small models.

The mistake is choosing a model once, at the start, when the prompt was hardest, and never revisiting it. Capability is per task, not per project. A workflow that needs a frontier model for one step usually does not need it for the other six.

Pick the model per step. Start at the bottom and move up only when a step measurably fails, rather than starting at the top and never coming down.

- [ ] List every model call in the workflow, not every project.
- [ ] For each one, try the smallest model in the family first.
- [ ] Move up a tier only where a graded sample actually fails.
- [ ] Re-check after any prompt rewrite: a better prompt often lets a smaller model succeed.

2 Test on a sample before running anything at scale

A bad prompt run once is a mistake. Run across 200,000 records it is an expense.

Before a batch job, an agent run, or a migration, run the same work on a small random sample and grade the output yourself. Twenty to fifty examples is usually enough to see whether the approach works, and it costs a rounding error against the full run.

Grade against written acceptance criteria you set before looking at the output. Deciding what "good" means after seeing the results is how a workflow gets run three times.

Keep the sample. When you change the prompt or the model, re-run the same sample and compare, so you know whether a change helped rather than assuming it did.

- [ ] Write acceptance criteria before the first run.
- [ ] Sample 20 to 50 real records, not invented ones.
- [ ] Grade the sample, then decide on the model.
- [ ] Re-run the same sample after every prompt or model change.
- [ ] Only then run the full job.

3 Spin: paying repeatedly for the same answer

Tokens that do not move the task forward.

Spin is any usage that does not advance the work: re-asking a question that was already answered adequately, regenerating output for taste rather than correctness, an agent re-reading the same files each turn, or a conversation that grows longer without getting closer to a decision. It is usually the largest source of waste in interactive use, and it is invisible if you only look at cost per request instead of cost per finished task.

It is worth naming because the fix is behavioral rather than technical. Nothing about the model changes; what changes is when you stop.

How to test for it. Measure tokens per completed task rather than per call, and watch the ratio of input to output tokens across a session: a conversation where input keeps climbing while useful output stays flat is spinning, because every turn resends the whole history. Count attempts to an accepted answer on a sample of real sessions. Ask how many of your longest sessions produced a proportionally better result than your median one; usually the honest answer is no.

How to reduce it. Decide the acceptance criteria before prompting, so ”good enough” is a fact rather than a feeling. Cap retries at two or three, then change something structural (a different model, a different decomposition, or a person) instead of re-rolling the same request. When a conversation has gone down a wrong path, start a fresh one with a better prompt rather than appending to the bad context, because every further turn re-sends that history and pays for it again. Prefer one specific request over three vague ones.

- [ ] Track tokens per completed task, not per request.
- [ ] Set a retry cap, and change approach when you hit it.
- [ ] Restart the context after a wrong turn instead of appending.
- [ ] Write down what a good answer looks like before asking.

4 Send less context, and send the stable part first

Input dominates most workloads, and most of it is repetition.

In multi-turn work, input usually dwarfs output, because every turn resends everything before it. Long documents pasted whole, entire files given to a coding tool, and conversations that never reset are the common causes.

Retrieve rather than stuff. Pulling the three relevant paragraphs out of a long document costs a fraction of sending the whole thing, and it usually produces a better answer because the model is not hunting for the relevant part.

Every major provider now discounts context that is reused, because a cache read skips most of the computation. That discount is only available if the reused part of your prompt is byte-identical and comes first, so put the stable material (instructions, schemas, reference text) at the front and the part that changes at the end. Check your provider usage report for cached input tokens to confirm you are getting the discount you think you are.

Bound your outputs too. Set an explicit maximum length, and ask for the format you actually need. A request that returns three fields as JSON is cheaper and more useful than one that returns three paragraphs you then have to parse.

- [ ] Retrieve the relevant passages instead of pasting whole documents.
- [ ] Put stable instructions first so they can be cached.
- [ ] Verify cached input tokens are appearing in your usage report.
- [ ] Set maximum output length, and request structured output.
- [ ] Start new conversations rather than letting one run indefinitely.

5 Batch what is not urgent

Latency you do not need is a discount you are not taking.

Most providers offer an asynchronous batch mode at a substantial discount against the same model, in exchange for a slower turnaround. Anything that is not waiting on a person (overnight enrichment, backfills, evaluation runs, bulk classification, report generation) belongs there.

The same reasoning applies to scheduling. Work that has to happen but does not have to happen now can be queued and run together, which also makes it far easier to measure.

- [ ] Sort your workloads into interactive and not-interactive.
- [ ] Move everything not-interactive to a batch endpoint.
- [ ] Confirm the discount appears on your bill.

6 Turn off effort you are not using

Extended reasoning is a capability, not a default.

Models that can spend extra computation before answering produce far more tokens when they do, and those tokens are usually billed at output rates. On genuinely hard problems that is money well spent. On extraction, formatting, routing, and lookups it is pure waste, and it can make the answer worse by inviting the model to overthink a simple instruction.

The same applies to agentic tooling. An agent that can search, read files, and run commands will do all three unless the task tells it not to. Scope the task and give it the context it needs up front rather than letting it discover the context one expensive step at a time.

- [ ] Reserve extended reasoning for tasks where a sample shows it helps.
- [ ] Give agents the files and constraints up front.
- [ ] Bound tool use: how many searches, how many turns.

7 Do not ask a model at all

The cheapest call is the one you do not make.

A surprising share of production AI usage replaces something deterministic that already worked. Validating a date format, matching an exact string, sorting a list, deduplicating records, or looking a value up in a table are all faster, cheaper, and more reliable in ordinary code.

Caching your own results counts too. If the same question is asked repeatedly with the same inputs, store the answer. Many workloads have a long tail of exact repeats that nobody looks for.

- [ ] Audit for calls that a rule, a regular expression, or a lookup could do.
- [ ] Cache answers for repeated identical inputs.
- [ ] Ask whether a human already knows the answer.

8 Measure it, or none of the above holds

Reduction that is not measured tends to reverse.

Pick one denominator that means something to your organization, such as tokens per resolved ticket, per processed document, or per active user, and track it over time. Absolute totals will rise as adoption grows, which makes them useless for telling efficiency from growth.

Review the largest consumers monthly. Usage concentrates: it is common for a small number of workflows to account for most of the tokens, and those few are where this guide pays off.

- [ ] Choose a per-unit-of-work denominator and track it monthly.
- [ ] Identify the top workloads by token volume.
- [ ] Re-run this checklist against those workloads first.

What this guide does not claim

These are general practices, not measurements of your systems, and the savings from any one of them depend entirely on your workload. Nothing here is a statement about environmental outcomes: reducing token usage reduces the energy and water associated with serving it, but neither we nor anyone outside the providers can tell you by exactly how much, which is the same reason every figure R3CKON publishes carries a stated range.

Reduce first. What remains is what our ledger is for.