10 ways your AI bill gets hijacked

The unbounded-consumption failures that turn an LLM feature into a runaway bill — and the one control that stops each.

smallestbusiness.com
stop your AI bill from getting hijacked

It is not hypothetical. One leaked key ran up **$82,000 on a Gemini account in 48 hours**. Sysdig documented **LLMjacking on AWS Bedrock at over $46,000/day**. A **$400/mo OpenAI bill became $67,000** after a key sat 11 days in a public repo. In June 2026, **282 apps** were caught shipping LLM keys in the client. The cost is decided by one thing: how long the meter runs before anyone notices. OWASP names this **LLM10:2025 — Unbounded Consumption**. Here is how it happens, and how to close it.

1 A key leaks into a repo, app, or log

Provider keys committed to git, shipped in client JS or a mobile app, or printed to logs — then scraped and abused within hours.

→ **Keys server-side only**; scope + rotate them; secret-scan repos + CI (`gitleaks`, `trufflehog`) and rotate on any exposure.

2 An "alert" that never stops the spend

A budget alert emails you *after* the money is gone. At machine speed, that's an invoice, not a brake.

→ **A global hard cap that STOPS serving at $X** — fail-closed, on the request path, not just a notification.

THE ONE THAT MATTERS MOST

3 One key or user drains the whole account

A single shared key means one compromised customer, script, or teammate can burn your entire balance.

→ **Per-user / per-key daily budgets** (`$/day`); hand out per-user **virtual keys**, never the master key.

4 An agent loops and bills on every turn

A tool-calling or retry loop with no ceiling re-queues itself and spends on each iteration until something crashes.

→ **A per-run breaker**: `max_steps`, a per-run token budget, a repeat-tool-call hash, and a fan-out cap — halt mid-run, not on the invoice.

5 Rate limits count requests, not tokens

A "100 requests/min" limit does nothing when each request asks for a 200k-token completion.

→ **Token-aware rate limits** (tokens/min), `max_tokens` on every call, and a per-user concurrency cap. This is OWASP's named defense.

6 A public endpoint = someone's free ChatGPT

An unauthenticated chatbot, demo, or "free trial" gets botted and your account pays for strangers' inference.

→ **Auth on every AI endpoint** — a real, revocable identity per call; topic-lock it and add abuse detection. Treat it like a payment endpoint.

7 Always calling the most expensive model

Routing every request to the frontier model — including the trivial ones — multiplies the bill for no gain.

→ **Default to a cheap model**, escalate only when the task needs it. And ask: does this call need a model, or a function the model could write once?

8 Paying twice for the same tokens

Re-sending an identical prompt or a huge static system prompt on every call, uncached, pays full price each time.

→ **Cache repeat calls and the system prompt** (Anthropic prompt caching / OpenAI cached input) — often the single biggest cut.

9 Injection turns your agent against you

Hostile text in a web page, email, or document tells your agent to loop, exfiltrate, or trigger paid tools.

→ Treat **retrieved / third-party content as untrusted data**, never instructions. **Least agency**: no privileged or paid action without a deterministic check in your code.

10 Free credits hide the real burn

Trial credits and no per-call attribution mean nobody sees the true run-rate until the credits run out and the bill is real.

→ **Route every call through one gateway**; log **tokens, $, identity, run_id** per call; know your **list-price** cost with credits stripped out.

THE ONE RULE UNDERNEATH ALL TEN

When anything is in doubt — **cap hit, breaker tripped, auth missing, abuse detected, budget spent** — stop or degrade; **never run unbounded**. Fail-open is what you inherit; **fail-closed is what you build**. Every gap above is a way the bill, or the liability, runs away on its own.

Start today: put one **fail-closed hard cap** on your main LLM route — the kind that **stops**, not the kind that just emails you. That closes #2, the one that turns a mistake into a catastrophe.

smallestbusiness.com