Where Did All My Credits Go?

Understanding and Reducing Token Usage in GitHub Copilot

Dr. Matthias Liebeck · Agentic Coding Summit #2 · August 17, 2026

◆ AGENDA

What we will cover

1	**The Problem**	what changed on June 1, and the claims flying around
2	**Token Anatomy**	how the harness works, what a turn really sends, where credits flow
3	**My Small Benchmark**	two real codebases, six real tasks, one honest method
4	**Techniques**	claimed vs. measured: models, context, compression
5	**Measure It Yourself**	one evening, no infrastructure, plus the observability handover
6	**Tomorrow & Conclusion**	six things to change tomorrow, three sentences to keep

◆ABOUT ME

About me

- Senior Solution Architect in Düsseldorf, focus on .NET, C# and Azure
- Software developer since 2009, C# since 2006
- 2009 – 2015: studied Computer Science & Mathematics
- 2018: PhD in Computer Science / Artificial Intelligence, focus on Natural Language Processing
- Organizer of the Azure Düsseldorf Meetup and this summit, speaker on Azure & agentic coding
- GitHub Copilot Newsletter: ghcp.liebeck.io

Azure Meetup Düsseldorf

Microsoft Certified: Azure Solutions Architect Expert

Microsoft Certified: DevOps Engineer Expert

Microsoft Certified: Azure Developer Associate

Microsoft Certified: Azure Administrator Associate

Microsoft Certified: Azure Data Scientist Associate

GitHub Copilot Certification Program

GitHub Copilot Newsletter

◆THE PROBLEM

June 1 changed the deal for GitHub Copilot

Before: premium requests

- Price per request, not per work done
- A short follow-up cost as much as an agent run of an hour, so people batched questions to save requests
- Some built endless loops that burned compute for a handful of requests. Flat rates invite exactly that, e.g., Ralph Wiggum loops
- And why not always pick the strongest model with the same multiplier? It cost the same

Since June 1: usage based

- AI credits (1 credit ≈ 1 cent), every token counts: input, output, cached
- Fairer for both sides: you pay for what you actually use
- The real problems now: high per-token prices and poor predictability of monthly costs
- What I see at: people hit limits, usage stops for days

◆ **Suddenly everyone asks the same question: where do my credits actually go?**

◆THE PROBLEM

Fairer, but nobody is happy yet

The mood on Reddit

> *I used to need half my monthly requests. Now my quota is gone after three days.*

The anger is real. But most of it is about not knowing where the credits go, and that part is fixable.

My personal take (opinion, not measurement)

- Copilot is simply the first established tool to bill this way. Claude's API has billed by tokens all along
- Much of today's AI tooling is subsidized for market share. The free ride will end
- Flat-rate-only offers will not survive at established providers
- So we better learn to price agentic coding into our own work, and pass the cost on to our customers

◆THE PROBLEM

The internet has answers. Big ones.

- “Caveman: ~65% less output”
- “Headroom: 15–20% less input”
- “Focus your context”
- “Pick a cheaper model”
- “Disable your MCP servers”
- “Compress your prompts”

Which of these would you bet money on?

My goal for the next 30 min

- Six things you can change tomorrow morning
- How to check any claim on this list yourself
- Numbers are Copilot, the method is tool-agnostic
- This list returns at the end, annotated with measurements
- 💡 marks the slides worth remembering. If you keep only those, you got the talk

02

Token Anatomy

Where credits actually flow

◆TOKEN ANATOMY

What happens when you press Enter

You

one prompt

The GitHub Copilot harness

assembles context (files, instructions, memories)
exposes tools: read, edit, shell, search, docs lookup, GitHub MCP
runs the agent loop: continue, retry, stop
manages caching and sub-agents

Model of your choice

GPT, Claude, Gemini, Kimi, MAI.
Swappable and “stateless”

The model is the interchangeable part. The harness is the constant: it decides what gets sent, which is what you pay for.

💡 **You are using (and paying for) a system, not a model.**

◆TOKEN ANATOMY

The model remembers nothing

Every single call re-sends the whole package. “Being in context” literally means “being in this payload”.

System prompt + your instructions	stable prefix, cheap cache reads after turn one
Tool definitions (~14k)	stable prefix, cached too
Entire conversation so far	grows every turn: your messages, answers, all tool results
Your new message	the only genuinely new part

Measured: context grows from ~21k before turn one to 90k–110k at session end. Re-sent every turn, that sums to 1.2–2.7M input tokens per task, 60k–90k per turn.

💡 **In context means re-sent, and re-billed, on every single call.**

◆TOKEN ANATOMY

Four token types, four prices

Input

everything sent to the model. the big one.

Output

what the model writes back. priciest per token.

Cache write

storing the baseline copy. billed extra on Anthropic models.

Cache read

reusing the stored copy. cheap, 5 min TTL.

How the prompt cache works

- The provider stores the stable prefix of your request (system prompt, tools, history) for 5 minutes. Writing that copy is billed once, re-reading it costs ~10% of a fresh send.

GitHub’s official taxonomy: input, output and cached tokens, priced per model, converted to AI credits at 1 credit = $0.01.

💡 **Every new session pays the baseline. Within a session, the cache pays you back.**

◆ TOKEN ANATOMY

The cache: who runs it, how long it lives

Where does it live?	On the provider’s servers, next to the model. Not on your machine. The harness just benefits from it.
Who sets the TTL?	The model provider. My runs report 300 seconds (cacheTtlSeconds in the session log). Every call inside the TTL refreshes the clock.
What exactly is cached?	The stable prefix of the request: system prompt, tool definitions, conversation so far. Reads cost ~10% of input. Anthropic models bill the write.
Resume tomorrow?	The cache is long gone, but nothing is lost: the transcript lives on your disk. Turn one re-sends the whole history as fresh input, and pays the cache write again.

◆ **The provider’s cache saves you money. Your disk saves the conversation. Only one survives the night.**

◆TOKEN ANATOMY

The 20,000-token OK

```
copilot -p "Answer only with OK"
```

19,752
input tokens

12.36
AI credits

68
tokens of actual conversation

The other 99.7%: the session baseline

- About two thirds tool definitions (including the built-in GitHub MCP server), one third system prompt. A whole repo with instructions and skills adds only +3%.

tool definitions ~14.1k | system prompt ~7.0k | conversation: 68

Measured Aug 2, 2026, CLI 1.0.75, gpt-5.6-sol, exact values from the session log. Spread across repeats: 2 to 12 tokens.

◆ **You pay for the whole stage before your four words enter it.**

◆TOKEN ANATOMY

Official prices per 1M tokens (USD)

Model	Input	Cached input	Cache write	Output
GPT-5.6 Sol	$5.00	$0.50	–	$30.00
GPT-5.6 Terra	$2.00	$0.20	–	$12.00
Claude Sonnet 5 (promo until Aug 31)	$2.00	$0.20	$2.50	$10.00
Claude Opus 5	$5.00	$0.50	$6.25	$25.00
Claude Fable 5	$10.00	$1.00	$12.50	$50.00
MAI-Code-1-Flash	$0.75	$0.075	–	$4.50
Grok 4.5	$2.00	$0.50	–	$6.00
Kimi K2.7 Code	$0.95	$0.19	–	$4.00

- Output costs 4 to 6 times input. Cached input is a fraction, but not a constant one: 10% at OpenAI and Anthropic, 20% at Kimi, 25% at Grok
- Anthropic models additionally bill cache writes. Tiered models double their rates past the long-context threshold (GPT-5.6 Sol: 2× past 272K input)

Source: GitHub docs, models and pricing, captured Aug 3, 2026. Default tiers shown. Prices move, remember the 90% promo.

◆ TOKEN ANATOMY

One prompt, a whole orchestra

Main LLM

plans and writes the code

Helper model

small side tasks

Search sub-agent

explores the repo

Embedding model

semantic lookup

How sub-agents fit in, and why they cost extra

- A sub-agent runs in its own session with its own baseline and an isolated context, then returns only a compact summary to the parent as a tool result
- The point: exploration tokens never pollute the parent context. The price: every helper pays its own way

◆ **All four instruments are on your bill.**

◆TOKEN ANATOMY

What a real task looks like

300k–1.2M

input tokens per run

>90%

of it cache reads

2k–10k

output tokens per run

Measured across 54 baseline runs on six real tasks.

💡 **86% of the credits buy input. Measured, not assumed. Most savings tips aim at the wrong end.**

Credit split, computed per run from token splits and per-direction rates: gpt 83%, kimi 91%, MAI 75% of credits are input.

◆TOKEN ANATOMY

The context window

The cost side (measured territory)

- Everything loaded is re-sent on every call: 500k vs. 200k means 2.5× input per turn
- Past the long-context threshold the rate itself doubles (GPT-5.6 Sol: 2× past 272K input, official pricing)
- Cache softens it, but writes are expensive and the cache lives 300 seconds
- Latency grows with payload size

The quality side (research, not my benchmark)

- Long-context models lose precision in the middle of the window (“lost in the middle”)
- Irrelevant material does not just cost money, it actively distracts (“context rot”)
- A big window is capacity, not comprehension

💡 **A context window is a budget, not a backpack you have to fill.**

03

My Small Benchmark

Two real codebases, six real tasks

◆ MY SMALL BENCHMARK

Not a toy repo. Two production systems.

gesellschaftsspieler-gesucht.de

- German board-game community, ~19,000 users
- ASP.NET Core MVC on .NET 9, EF Core, Identity, SignalR, Hangfire
- Years in production, grown conventions

SmartWarten

- Digital waiting queue SaaS, my own company
- ASP.NET Core, multi-tenant, feature flags, background jobs
- production code as well

Where do the headline numbers come from?

- SWE-bench Verified: 500 real GitHub issues, all from open-source Python repos. The benchmark the latest Claude, GPT and Gemini releases lead with (Opus 4.5: 80.9%). HumanEval: Python functions from docstrings.
- My daily work is C# in grown codebases, so that is what I measure on.

◆ MY SMALL BENCHMARK

Six tasks from my own code

I wanted to know what these techniques do to my code, not to a lab-grown experimental setup. So I picked tasks from two repositories with a similar coding style.

Gesellschaftsspieler-gesucht
IsLocked bugfix: locked meetups still visible and announced
Announcer job: only mail users active in the last 2 years
Duplicate sidebar: prompt names 1 view, real fix touched 6

SmartWarten
Auto-enable feature flag on demo self-signup
Hide category dropdown when only one category exists
Create SLA and general terms PDFs as tenant documents (--attachment)

All six were already solved once by a human. The real fix provides the pass checklist.

💡 **Measure on your own task mix. That is the whole point.**

◆ MY SMALL BENCHMARK

How I tested each run

1. Check out the git commit from just before the human fix
2. Load the frozen prompt from a file, identical for every run
3. Invoke the Copilot CLI with a pinned model, the repo path and the prompt
4. Record the consumption: tokens and AI credits
5. Archive everything into one folder per run: stdout, session log, git diff
6. Judge the git diff manually against the human reference fix

Then: git reset --hard, next run.

◆ **Same start, same prompt, same model. Only then are runs comparable.**

◆ MY SMALL BENCHMARK

How I judged quality

Cost without quality is half the story. Every run was judged, in three stages.

1 Pass criteria written before the runs, derived from my own reference fix. I had solved every task myself once

2 Automated diff review by an LLM, against my reference commit

3 Manual review of every diff. Because the automated judge was wrong twice before I pushed back

Where automation fooled me

- First automated verdict: “no run wrote real PDFs”. Spectacularly wrong, and wrong in the direction that fits the thesis
- Cause: Git for Windows renders PDFs as text in diffs. The truth was in the blob hash. Check hash and size, not the rendering

What the reviews caught

- The right fix on the wrong feature: my app selects issues in two places, walk-in queue and appointment booking. The task was the queue. Several runs cleanly fixed the booking. Flagged by the LLM review and confirmed manually
- A frontend dropdown bug that compiles fine and only breaks in the running app

Benchmarking yourself? Write the tests before the agent starts. Never let it write the test it will pass.

💡 **The dangerous failures compile. The build is not your judge.**

◆ MY SMALL BENCHMARK

What did running this cost?

157

runs across 10 experiments

6,507

AI credits, all series

~$65

about 60 EUR, failures included

- 7 evenings of setup and evaluation.
- The whole benchmark cost less than a conference ticket.

◆ MY SMALL BENCHMARK

The honesty slide

> *“My tests don’t merit the accuracy of an academic paper.*
> *Each experiment ran three times, so variance and LLM stability factor in.*
>
> *Don’t quote me on exact numbers.*
>
> *Take home instead: Matthias found X, Y and Z on his codebase, and today I learned how to measure this on my own codebase.”*

- Median and range, never mean. Aborted runs reported separately, their cost stays in the bill.
- n=3 is too small for the spread I saw (same model, same approach, one line of difference, 1 of 3 passed). More runs were simply too expensive for me.
- Building a large public benchmark is genuinely hard. A one-evening check on your own repo is not. Do the second one.
- **I measure cost, not value**. Whether AI makes you productive is a harder, separate question. This talk keeps you from overpaying, it does not tell you what is worth building.

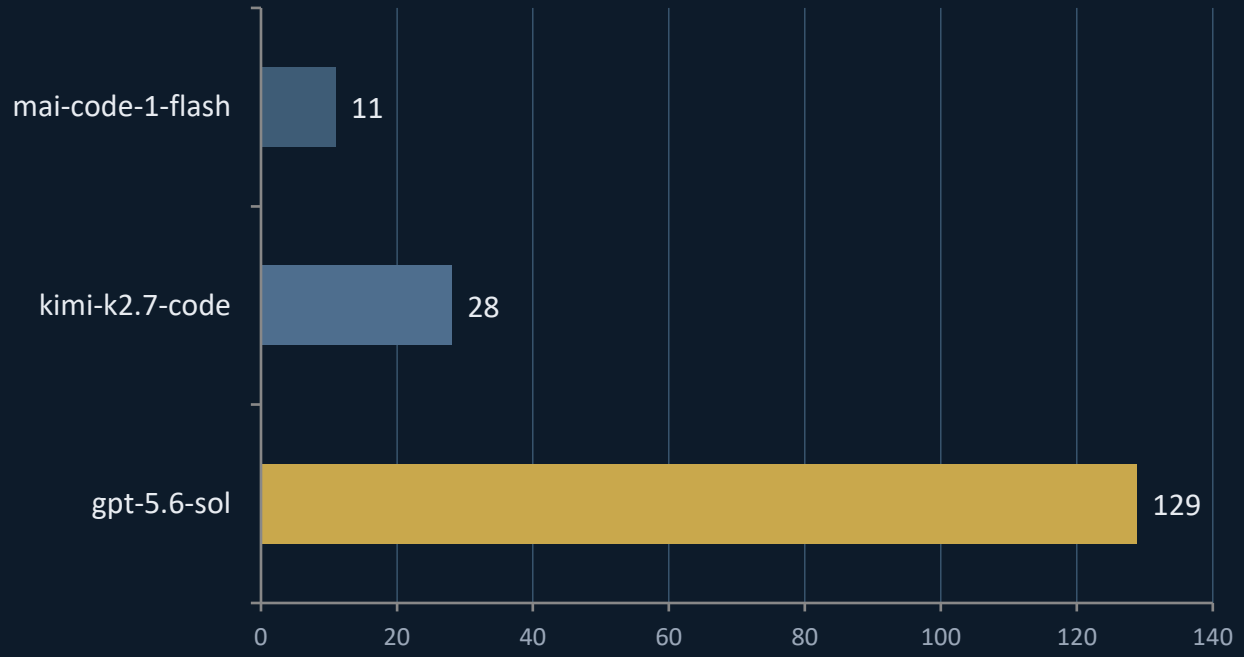
04

Techniques

Claimed vs. measured

INPUT OUTPUT

Same task, up to factor 12 in credits



Scorecard rules, and a warning

- Cost per successful task, failed runs stay in the denominator
- At unified prices across six tasks: gpt vs. kimi 3.8–5.4×, gpt vs. MAI 2.2–11.8×. The lever is real, but task-dependent
- Chart: exp02, Jul 27, all models at full price. Median and min–max per task in the data
- Capability limits: MAI reads no PDFs and no images, Kimi has no effort flag
- Prices move under you: my cheapest model ran a 90% promo until Aug 2. I had to re-measure its credits. Token counts stayed stable, so measuring in tokens is stabler than in credits
- Pass rates, all six tasks judged: three passed across the board, two nearly so. The one task that separates: gpt 6/6, kimi 5/6, MAI 2/6. **Cheap gets expensive when it fails**

💡 **Model rankings expire. Build a measuring habit, not a bookmark to a leaderboard. Token costs can change.**

The input side is where the money is

Focus your context	Fresh session = pay the ~20k baseline again. Long session = cheap reads, growing context. A real trade-off.
Instructions hygiene	Instruction files are cheap at rest (my repo adds only +3%), expensive in what they trigger: one line commanding “read all memory-bank files” costs thousands of tokens per task.
Tool and MCP hygiene	The biggest fixed block: ~14k tokens of tool definitions per session, measured, and that is with only the default servers. Disable what you don’t need.
Don’t break the cache	A model switch mid-session turns cheap cache reads back into expensive writes.

◆ **Everything loaded into your context is billed, every session.**

What does a screenshot cost?

Same .NET stacktrace, two forms: a 94 KB screenshot vs. 808 characters of plain text. Trivial prompt, so the delta is exactly the attachment.

As screenshot (PNG)

+1,188

input tokens (gpt-5.6-sol; kimi: +1,333). Credits: +0.74

Same content as text

+179

input tokens. Credits: +0.11. Factor 6.4× (kimi: 7.1×)

- File size says nothing: the PNG is 117× larger as a file, but only 6.4× in tokens. Images map to a fixed tile grid
- MAI cannot read images at all: clean abort, 0 credits. Capability limits cost nothing if detected cleanly

◆ **Paste text, not screenshots**

◆TECHNIQUES · INPUT SIDE

How not to measure input tokens for something small

My first attempt used a real coding task with the stacktrace attached. 27 runs, 852 credits. The result was unusable.

The noise

- Identical runs of one variant: 332k, 585k, 1,106k input tokens
- Measured deltas: +145k, +873k, −95k
- The real attachment effect: ~1,200 tokens. Noise was 100 to 800 times the signal

The fix

- Switch off the dominant variance source: agentic exploration
- A trivial prompt (“Answer only with OK”) makes the run deterministic, spread 2 to 16 tokens
- The delta to the control group is then exactly the attachment cost

◆ **To measure a small number, switch off the big noise first.**

How Caveman works

An instructions file that tells the agent to answer like a caveman: no filler, no hedging, no politeness.

Without Caveman

“I’ll now carefully update the QueueAllowance service so that demo queues are included in the count. Let me first examine the existing implementation to understand…”

With Caveman

“Fix QueueAllowance. Demo queues count now.”

- In my setup: injected as AGENTS.md into the repo per run, verified in the system prompt
- Touches only natural-language output. Code and tool calls stay untouched, by design

◆ **Caveman targets output tokens. Remember which token type that is.**

Caveman: claimed 65%, measured −22% and +70%

Claimed

−65%

output tokens, per the README

Measured on gpt

−22%

credits, but via less input and shorter turns, barely via output

Measured on kimi

+70%

kimi obeyed the style faithfully (“Need inspect project layout and memory first.”) and still cost more: it simply worked longer, +50% input tokens on one task

Structural reason: agentic-coding output is code and tool calls. There is nothing chatty to compress.

💡 **A savings technique without a measurement is a rumor.**

How Headroom works

A local proxy that sits between the CLI and the model API and compresses the request on the way through.

Routing via provider URL environment variable, pointed at localhost. Set it only in the measuring shell.

- Targets input tokens, the dominant cost in my measurements
- The risk to watch: rewriting content can invalidate cache prefixes, cheap cache reads become expensive writes

◆ **Fewer input tokens is not automatically fewer credits. The cache has a say.**

Headroom: claimed 15–20%. Measured: it did not run.

- I was unable to get it working after two evenings of trying, due to authentication issues between the proxy and Copilot
- Fair context: Headroom marks its Copilot subscription mode as experimental, and the docs list only macOS as tested. I was on Windows

◆ **A negative result is still a result. Finding out cost a few credits. Maybe Headroom will work for GitHub Copilot in the future.**

◆TECHNIQUES · COMMUNITY

token-economy: the four quick wins

AJ Enns https://github.com/aj-enns/token-economy

A practical UBB guide. Its own “if you only do five things” list:

1. Use the cheapest model that meets the bar. Per-token prices span roughly 25× across the lineup
2. Ask narrow questions to keep output short. Output is priced 4–8× input
3. Start a new chat when topics change. Old turns get re-sent as input on every reply
4. Skip Chat for trivial code. Inline and next edit suggestions cost no AI credits on paid plans

◆ TECHNIQUES · COMMUNITY

github-copilot-token-optimization: 40+ techniques

Marco Olivo https://github.com/olivomarco/github-copilot-token-optimization

A field guide with a technique matrix, quality impact ratings and a diminishing-returns curve. Highlights:

Habits

- Close editor tabs you are not working on, the #1 source of auto-included IDE context
- Short output as project default (“Code only. No explanations unless asked.”), claimed 30–60% output savings
- Plan with a strong model, execute the saved plan in a fresh session with a cheaper one
- Convert Office and PDF inputs to Markdown before AI work

See also: xebia.com/articles/sensible-usage-of-your-tokens (Rob Bos)

◆ **The community catalog is rich. Treat every claim on it as a hypothesis to test.**

05

Measure It Yourself

One evening, no infrastructure

◆ MEASURE IT YOURSELF

The CLI is the measuring bench

```
copilot -p "<prompt>" --model <m> --session-id <guid> -C <repo> --allow-all-tools --no-auto-update
--max-ai-credits 500
```

- Per run: git checkout frozen commit → run → collect → git reset → next. A whole series runs unattended.
- Pin the model and effort (defaults drift, I watched mine change). The CLI version cannot be pinned, but --no-auto-update keeps it constant across a series. Log it per run.
- --max-ai-credits is your airbag per run.

◆ **Scriptable end to end. Write your pass criteria before the runs.**

◆ MEASURE IT YOURSELF

Where the numbers live

stdout, for free

```
AI Credits 12.36 (4s)
Tokens ↑ 19.8k (19.8k written) • ↓ 5
```

Credits plus all four token types, parseable with one regex.

events.jsonl, exact

```
~/.copilot/session-state/<id>/events.jsonl
```

One directory per session: credits to 9 decimals (totalNanoAiu), token split, duration, model, CLI version, git commit. The session archives itself.

◆ **No OTel, no dashboards needed. Two files hold everything.**

◆ MEASURE IT YOURSELF

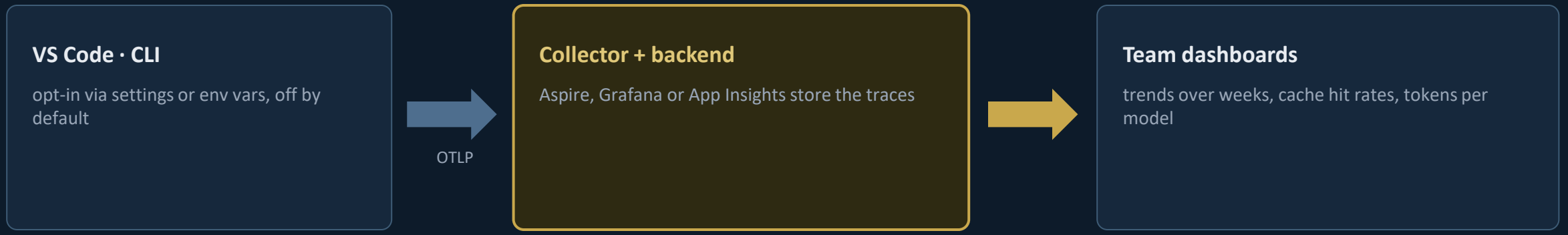
The claims list, annotated

“Caveman: ~65% less output”	measured by me: −22% / +70%, model-dependent
“Headroom: 15–20% less input”	did not run on Windows + CLI 1.0.75
“Focus your context”	measured by me: input dominates, every fresh session pays the ~20k baseline again
“Pick a cheaper model”	measured by me: factor 5–12, but rankings expire

◆ MEASURE IT YOURSELF

Beyond one machine: OpenTelemetry

Copilot can emit standardized telemetry: one trace per agent run, one span per LLM call, with token counts, duration and model.



Coverage today: VS Code yes · CLI yes · desktop app no (open feature request github/app#133, tested myself) · credits hide well: a field named cost says 0, the real value ships as nanoAiu

◆ **Up next: Maxim shows you the observability side.**

06

Tomorrow & Conclusion

What you take home

◆ TOMORROW

Six things you can do tomorrow

1. Small task -> small model (or auto). The big model is for the hard 20%.
2. Paste text, not screenshots. If it must be an image, crop it.
3. Audit MCP servers and instructions regularly. Everything loaded is billed every session.
4. Don’t switch models mid-session. It turns cheap cache reads into expensive writes.
5. New task, new session, but batch related questions. The baseline amortizes.
6. Benchmarking yourself? Write the tests before the agent starts. Never let it write the test it will pass.

💡 **You are not at the mercy of the counter. You can see where it goes, and steer it.**

◆ CONCLUSION

Three sentences to remember

1 Your credits buy input: 86% of them, measured across all models and tasks. The session baseline, the context, the helper models. Output looks expensive per token, and it is. But never forget how much input you send along with every single call.

2 Model choice is the biggest lever, and a moving target. Measure on your own tasks, prices change under your feet.

3 Trust no savings claim, including mine. The same technique saved 22% on one model and cost 70% extra on another. Your task mix decides.

One number you can measure this week: what does your next “say hello” cost?

Newsletter: ghcp.liebeck.io

Thank You!

Questions go to the roundtable at 13:55. Everything else: scan me.

Connect on LinkedIn

linkedin.com/in/matthias-liebeck

GitHub Copilot Newsletter

ghcp.liebeck.io