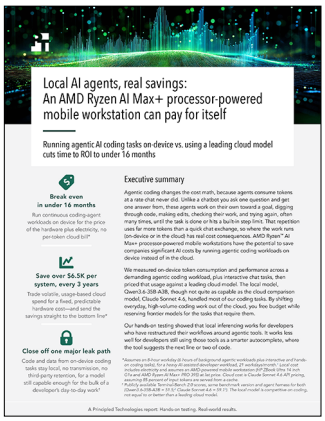


The science behind the report:

Local AI agents, real savings: An AMD Ryzen AI Max+ processor-powered mobile workstation can pay for itself

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Local AI agents, real savings: An AMD Ryzen AI Max+ processor-powered mobile workstation can pay for itself.

We concluded our hands-on testing on July 14, 2026. Token counts and throughput come directly from the model's own reported usage, captured at the HTTP boundary by a logging proxy; we never re-tokenized or estimated. All cost figures are computed from those measured token volumes and the stated price and power inputs. Prices are current as of July 2026 and may change.

Our results

To learn more about how we have calculated the wins in this report, go to https://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

We measured a representative AI-assisted developer workload, a RAG coding agent, on a single AMD Ryzen™ AI Max+ PRO 395 processor-based mobile workstation. We also measured interactive chat tasks and raw throughput probes on the same hardware for comparison. All measurements ran at concurrency 1.

We used a compressed (quantized) model for these measurements. The public coding score we report for Qwen3.6-35B-A3B (51.5 on Terminal-Bench 2.0) is a separate benchmark result for the full-precision version of the model, not a measurement from this harness. Full-precision models tend to score slightly higher than quantized builds, though the difference varies by task.

Table 1: Chat vs. RAG agent per-task token cost. The 78 runs are 78 executions of the same 5 pinned tasks (real fixes and features in Python-Markdown 3.7, each gated by that project's own test suite). The 96 percent is 27 of 28 background executions that passed their gated tests. The one that did not pass does not indicate the model is incapable of the work. That task was the hardest of the five tasks, a one-character fix in a roughly 989-line core file, and the same task passed on 5 of its 6 attempts (per PT's internal test logs for this project). On that single run the agent ran out of the step limit we imposed (a cap of 14 actions per job), before finding a solution, so it stopped on our constraint rather than because the model couldn't do it. There is no internal baseline for the 96 percent itself, because we did not run a cloud model through the same harness on these 5 tasks.

Task type	Mean tokens	Median	Range	Runs
Interactive chat task	2,640	2,517	2,315 - 3,623	9
Agentic task (RAG)	138,769	57,049	22,750 - 658,670	78 (50 foreground + 28 background)

Table 2: Raw model speed and sustained agent rate throughput. Measured in tokens per second (tok/s) and tokens per hour (tok/hr). Scripted sustained throughput is the billing-token rate of a continuous scripted workload with a balanced input/output mix (about 7:1), so it is limited by generation (decode) speed. Background agent rate is the billing-token rate of the RAG coding agent working real jobs back to back; agentic coding is input-dominated (about 71:1), and input is processed as prefill several times faster than output is generated, so it accumulates billing tokens roughly 1.8 times faster while producing less output. The two differ because they measure different work.

Measurement	Result	Runs
Single-stream decode (short context)	54.69 tok/s	-
Decode at 128K context	33.89 tok/s (sd 0.04)	3
Prefill at ~10K input	681.3 tok/s	3
Prefill at 128K context	332.2 tok/s	3
Scripted sustained throughput	1,418,571 billing tok/hr	1 (60 min)
Background agent rate	2,609,025 billing tok/hr	3 x 30-min blocks

Table 3: Autonomous agent completion (RAG coding agent, background runs).

Metric	Value
Completed workflows per hour	17.78
Jobs passed (background)	27 of 28 (96%)
Average time per completed workflow	~3.4 minutes
Tokens per completed workflow	~146,700 (~144,700 input, ~2,000 output)

Table 4: RAG agent CPU and GPU utilization by workflow stage. The retrieval and indexing work runs on the CPU; the model generation runs on the GPU.

Stage	Share of time	CPU	GPU
Indexing and retrieval	~35%	~50%	idle
Generation	~65%	~3%	~90%
Whole run (average)	-	~20%	~58% (100% peak)

Projected daily token volume

We modeled a heavy AI-assisted developer's day as foreground plus background work: in the foreground, 30 interactive chat tasks and 4 hands-on agentic tasks; in the background, 6 hours of the agent running autonomously. The table below shows how these sum to the projected daily volume.

Table 5: Projected daily token volume (heavy AI-assisted developer workload).

Component	Assumption	Tokens/day
Foreground (interactive)	30 chat + 4 hands-on agentic tasks	634,263
Background (autonomous)	6 agent-hours at the measured background rate	15,654,154
Total projected		16,288,417 (16,001,604 input / 286,813 output)

Our cost assessment

We turned the measured daily token volume into a local-versus-cloud cost comparison. The token volume is measured; the prices, power, and usage intensity are stated inputs.

Costs and inputs we used:

- **Cost of hardware:** $5,080 HP list price for the ZBook Ultra G1a, before tax and shipping. Your price may vary by configuration and seller.
- **Cost of energy:** $0.15 per kWh, broadly aligned with the U.S. Energy Information Administration average retail electricity prices. System power modeled at 140 watts, the ceiling set by the laptop's 140-watt USB-C adapter; the processor's default TDP is 55 watts (configurable to 120). At 8 hours/day and 21 workdays/month this is about $3.53/month, and the real draw is likely lower.
- **Cloud comparison:** Claude Sonnet 4.6 at API list price ($3.00/M input, $15.00/M output, $0.30/M cached input). We assume 85% of input is served from cache, near the low end of the 84-99% cache-hit range measured for real coding agents.

We did not include IT labor, networking, cooling, software licensing, or cloud-management overhead on either side. Hardware is treated as a one-time capital cost; the cloud API is priced per token.

Table 6: Cost assessment of heavy AI-assisted developer workload results (6 background agent-hours/day, 85 percent cache).

Metric	Value
Cloud cost avoided per month	$327.25
Local electricity per month	$3.53
Net savings per month	$323.72
Payback (hardware / net monthly savings)	15.7 months
On-device (hardware and power) cost over 3 years	$5,207
Cloud spend over 3 years	$11,781
3-year savings	$6,574
Cloud-equivalent cost per completed workflow	$0.13

How we calculated the payback

- Daily foreground = (chat tasks/day x chat task cost) + (agentic tasks/day x agentic task cost).
- Daily background = background agent-hours/day x measured background-agent billing rate.
- Daily total = foreground + background; split into input and output at the measured ratios.
- Cloud cost/day = (fresh input x $3/M) + (cached input x $0.30/M) + (output x $15/M), where cached input = 85% of input. Cloud cost/month = cost/day x 21 workdays.
- Net savings/month = cloud cost/month - local electricity/month. Payback (months) = hardware price / net savings/month. Three-year savings = (cloud/month x 36) - (hardware + electricity/month x 36).

System configuration information

Table 7: Detailed information on the mobile workstation we tested.

Component	Detail
System	HP ZBook Ultra 14 inch G1a mobile workstation
System BIOS	X89 Ver. 01.05.07 (05/05/2026)
BIOS graphics memory setting	512 MB
Processor	AMD Ryzen AI Max+ PRO 395
Graphics	Integrated AMD Radeon 8060S
Memory	128 GB unified LPDDR5x
Storage	1 TB SSD
Operating system	Windows 11 Pro 25H2
Power adapter	140 W USB-C

How we tested

We built a token-economics harness that drove a local, OpenAI-compatible inference server through a logging proxy. Every request and response passed through the proxy, and token counts are read verbatim from the model's own usage field. Every call was streamed, so prompt processing (time to first token) and steady-state generation are timed separately. Per-task and sustained measurements ran one request at a time.

The AI coding agent

The workload was a RAG system built on a local, optimized embedding pipeline. It indexed a pinned real codebase into a local vector index, retrieved the most relevant code at each step, and chose actions from a toolset (search, read a file, edit a file, run the tests). The agent iterated until the project's own test suite passed or it reached a step limit.

Success was determined solely by test results. Failed runs stayed in the averages, so reported metrics reflect real-world performance, not best-case outcomes.

Embeddings for the index ran on the CPU, using nomic-embed-text-v1.5, via fastembed (ONNX Runtime backend).

The codebase under test was Python-Markdown 3.7, pinned and vendored so every run started from an identical baseline. Each task ran against a materialized (defective) version of the codebase.

The agentic coding tasks

The agent ran five pinned foreground tasks, each a real fix or feature in the Python-Markdown codebase, with a gated test suite deciding success:

Table 8: Agentic (foreground) tasks.

Task	What it exercises
tables_center_align	Center-alignment handling in the tables extension; multi-step retrieval and edit.
link_title_quote	Link-title quoting, the hardest task: a one-character fix in a ~989-line core file.
multifile_merge	A change spanning two files; cross-file reasoning and edits.
toc_slug_lowercase	Lowercasing of table-of-contents slugs in the TOC extension.
abbr_longest_match	Longest-match behavior in the abbreviations extension.

- **Background measurement** drove the same agent through a queue of these jobs back to back, across three 30-minute continuous blocks, to measure the sustained autonomous rate and the completed-workflows-per-hour.
- **Interactive chat scenarios** (code review, bug fix, test generation) measured the chat per-task cost in a chat setting, rather than an autonomous agent loop.
- **A padded-context agent loop** (a three-bug pricing fix and a module refactor) simulated agents that re-send full context for every step. This was retained as an upper-bound sensitivity case, not a typical-use estimate.
- **Two throughput probes** isolated raw model speed, independent of the agent framework: a ~10,000-token prefill probe with a cache-defeating marker, and a 128K large-context probe.

What counts as a billing token

A billing token is any token a cloud provider would meter and charge for: every token in the request sent to the model (the prompt, or input tokens) plus every token the model generates in reply (the completion, or output tokens).

We report these counts exactly as the model's own inference server returns them in the usage field of each response (prompt_tokens and completion_tokens), captured at the HTTP boundary by a logging proxy. We never re-tokenized text or estimated counts ourselves, so the figures match what the model actually processed and what a cloud API would bill.

In agentic coding, the input side dominates: each step re-sends the growing context, the retrieved code, and prior tool output, while the model writes back only a small edit. The background agent alone runs about 71 input tokens for every output token; blended with the day's chat tasks, the projected daily total works out to roughly 56 to 1.

Because cloud pricing bills input and output at different rates, and discounts repeated input as cached, we split every measured total into input and output before pricing it.

Measurement definitions

- **Decode rate** = completion tokens / (total request time - time to first token).
- **Prefill rate** = prompt tokens / time to first token on a near-zero-output request.
- **Sustained and background rates** = total billing tokens / elapsed working time, scaled to an hour.
- **Utilization by stage** = per-second CPU/GPU samples correlated with the workflow's phase markers.

Sample sizes and reproducibility

The workload is pinned and deterministic; the only run-to-run variation comes from the model's non-zero sampling temperature, so headline figures carry a distribution (median, range, run count). We completed three full runs and report the median run.

Software versions we used

Table 9: Software versions we used.

Component	Version / detail
Operating system	Windows 11 Pro 25H2
Inference runtime	LM Studio 0.4.16
Inference engine	llama.cpp (Vulkan backend, Windows) v2.25.2
Graphics driver	AMD Software: Adrenalin Edition 25.20.18 (06/20/2026); provides the Vulkan runtime the inference engine uses
Language model	Qwen3.6-35B-A3B, Q4_K_M quantization (lmstudio-community GGUF)
Embedding model	nomic-embed-text-v1.5 (fastembed / onnxruntime, CPU)
Corpus under test	Python-Markdown 3.7 (pinned)
Benchmark harness	Token-economics harness v2.0.1 (Principled Technologies)

Table 10: Inference settings (LM Studio 0.4.16).

Setting	Value
Context length	190,000 tokens (model maximum 262,144)
GPU offload	40 of 40 layers (full)
CPU thread pool size	12
Evaluation batch size	2,048
Physical batch size	512
Max concurrent predictions	6
Unified KV cache	On
Offload KV cache to GPU memory	On
Keep model in memory	On
Flash attention	On
K / V cache quantization	Off (full-precision KV cache)
Number of experts (activated per token)	8
Force MoE weights onto CPU	0 layers
mmap()	Off
Seed	Random for study runs; a fixed seed reproduces a run exactly

Sources for assumptions

- System pricing (hardware list price): HP ZBook Ultra configuration page, accessed June 19, 2026. https://www.hp.com/us-en/workstations/zbook-ultra-configure.html.
- Electricity rate: U.S. EIA Electric Power Monthly, Table 5.6.A, average retail price by end-use sector (April 2026 U.S.: 13.88¢/kWh all-sector, 18.83¢/kWh residential); $0.15/kWh used as a round figure. https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_6_a.
- Usage intensity (heavy AI-assisted developer, agent hours): Anthropic, "How Claude Code is used in practice" (heavy users average ~20 hrs/week of agent engagement). https://www.anthropic.com/research/claude-code-expertise.
- Cache-hit rate and input-to-output ratio: "Agentic AI Workload Characteristics," arXiv:2605.26297 (measured cache-hit 84-99%, input:output 54-560x). https://arxiv.org/abs/2605.26297.
- Coding quality (Terminal-Bench 2.0, publicly available): Qwen3.6-35B-A3B 51.5 (https://huggingface.co/Qwen/Qwen3.6-35B-A3B); Claude Sonnet 4.6 59.1 (https://www.anthropic.com/news/claude-sonnet-4-6).
- Cloud API pricing (Claude Sonnet 4.6 list): https://platform.claude.com/docs/en/about-claude/pricing.
- System power / rated wattage: HP ZBook Ultra 14 inch G1a QuickSpecs (https://h20195.www2.hp.com/v2/GetDocument.aspx?docname=c09119722); AMD Ryzen AI Max+ PRO 395 (https://www.amd.com/en/products/processors/laptop/ryzen-pro/ai-max-pro-300-series/amd-ryzen-ai-max-plus-pro-395.html).

This project was commissioned by AMD.

Read the report ▶

Primary contributors

Tech: Duvan Izquierdo

Writing: Ticia I.

Design: Emily B.

PM: Trent W.

How we created this report

A PT team, which includes the contributors we've listed and others, created this report and performed the technical work behind it. We used AI to generate progress reports, draft a report outline, compare test documentation against report verbiage, and create the methodologies. The stock photo is also AI generated.

Principled Technologies®

Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc. All other product names are the trademarks of their respective owners. For additional information, review the science behind this report.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:

Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.