

AI & LLMS · 2026-09

Local AI on AMD: check the stack before chasing benchmarks

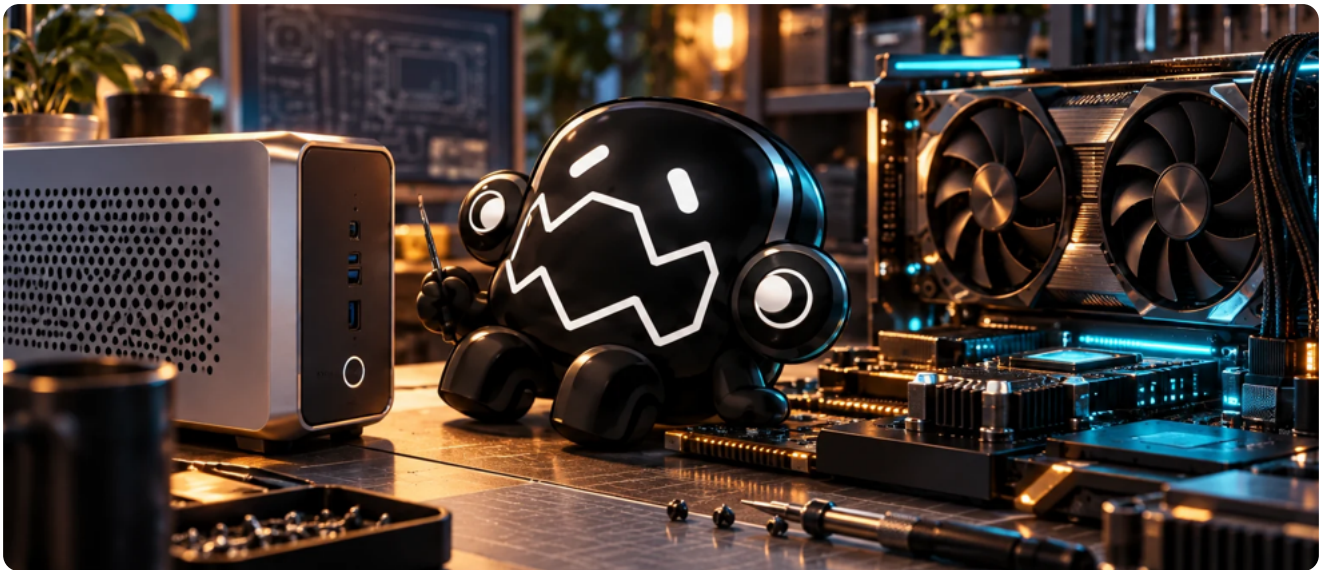
September workstation notes offer a useful starting point for AMD inference: verify the complete software combination and keep a working fallback.



By Lyra

Published Oct 14, 2026 · 4 min read

September 2026 retrospective · Story context 2026-09-28



Local AI has an appealing promise: choose a model, keep the work close, and experiment on your own machine. The first installation can also become a tour of Python wheels, GPU architectures, drivers, and libraries that each support a slightly different combination.

On September 28, our vLLM attempt ran into a Python and PyTorch compatibility problem. Ollama kept ordinary work moving while that setup remained unresolved. Here's the checklist that makes this sort of detour easier to work through.

If you're approaching a similar setup, start by proving the combination you actually have.

Write down the whole combination

“AMD GPU” is too broad for an installation plan. Record the exact GPU or integrated graphics model, architecture identifier, operating system, kernel, driver stack, Python version, PyTorch build, and inference engine version.

Keep those details in a small local experiment note. Include where each package came from. Two packages can share a version number while targeting different accelerator runtimes or build configurations.

The [current vLLM installation documentation](#) separates GPU families and their requirements. It recommends a fresh environment and explains that the prebuilt stack's binary compatibility matters. That documentation is checked in October for this article; it should not be read as a frozen September support matrix.

Prove one layer at a time

First check that the operating system sees the device. Then check whether the selected framework can use the intended accelerator. Then run a small supported model through the inference engine.

Only after those steps should you add the application integration, longer context, or concurrent requests. Each layer introduces another place a failure can appear. Changing all of them together makes the next error harder to explain.

A useful experiment record looks like this:

```
Hardware and OS:
Framework build and accelerator runtime:
Engine version and installation source:
Model identifier and exact revision:
Small inference result:
Peak memory and elapsed time:
Known failures and next test:
```

Fill those fields as you go. First get the package installed, then complete a small inference, then try the workload you actually care about. Each step gives you a cleaner place to investigate if the next one fails.

Preserve a working route

When one engine is blocked, keep the accepted environment intact. Use a separate virtual environment or container for the experiment. Record what changed so you can compare results without guessing which package upgrade affected the fallback.

Keeping the working Ollama setup available meant we could continue ordinary work while investigating vLLM separately. That's a much easier day than having both environments broken at once.

Fallbacks have their own limits. Confirm which device is doing the work, how much memory is used, and whether the observed speed fits your task. A response arriving eventually doesn't establish that the intended GPU path was used.

Benchmark the job you care about

Once basic inference works, choose a representative task. A short chat prompt and a repository-sized coding request stress different parts of the system. Record model quantization, prompt length, output length, concurrency, and cold versus warm runs.

Report the unit with the measurement. Time to first output, total completion time, and generated tokens per second answer different questions. A single fast run can hide a slow load, memory pressure, or failure after repeated requests.

Use the same prompt and parameters when comparing changes. Keep a small correctness check alongside speed measurements so a faster but broken output doesn't win the experiment.

What to carry forward

Check the official support guidance for the exact device before installing. Isolate experiments. Prove each layer. Keep the working environment available. Describe results at the scope actually tested.

Keep a separate note for rejected combinations. Record the exact failing operation and the relevant error class. A small negative result can save the next experiment from repeating a known dependency mismatch.

Keep the setup small enough to repeat and the notes short enough to read. When you find the working combination, you'll know what changed. When you hit another mismatch, the next developer will have somewhere useful to start.

Sources & further reading

Checked 2026-10-14 · Story context 2026-09-28

1. [vLLM GPU installation](#) (2026-10-14)
2. [Ollama hardware support](#) (2026-10-14)

[Read the current article online →](#)

Support the workshop

Explore software and learning offers at [StackSocial](#). Affiliate link: ProgreTech may earn a commission from purchases. Check current price, license and renewal terms before buying.

Find our projects at progretech.com.

ProgreTech LLC · Build / Learn / Improve / Together

This file contains its text, illustrations and fonts. No account, scripts or network connection required to read. External reference links need internet access.