

I've Got 99 Problems, but LLMs Ain't One!

Cedric Clyburn · Senior Developer Advocate, Red Hat



Why private and local AI?

83% of organizations are moving workloads back to private cloud or on-prem for privacy and cost ([Barclays CIO Survey 2024](#)).

Control & Privacy

Keep data on your infra, control updates and versions

Cost Predictability

Avoid unpredictable egress/inference bills

Customization

Tune models, add guardrails, ship your cadence

Portability

Run on Linux, Kubernetes, or bare metal

Let's get started!

About me

Cedric Clyburn

Senior Developer

Advocate, Red Hat



Organizer, [KCD New York](#)

Open source +

community builder ·

[@cedricclyburn](#)

Technical Educator

- [RAG vs Fine-Tuning](#)
- [Ollama](#)
- [vLLM](#)

Agenda

Open models: what's new in OSS

☹️ Why hosted APIs aren't always enough

⚙️ Private & sovereign AI options

☰ How to deploy: local → production

🔧 Tooling: llama.cpp, vLLM, TGI

📊 Cut GPU cost by ~50%: quantization

Open Source Model Explosion

Foundation models

Llama 3.x, Qwen, DeepSeek, Mistral, Gemma

instruct

base

MoE

Modalities

Text, vision, audio, diffusion, multi-modal

VLM (Llava, Ldefics)

ASR/TTS

Diffusion

Ecosystem

Tokenizers, safetensors, GGUF, serving runtimes

llama.cpp

vLLM

TGI

GGUF

safetensors

The weights are accessible... but how do we deploy, serve, and scale them responsibly? Think data gravity, GPUs/NUMA, batching, safety, updates, and SLOs.

Options to use LLMs today

Model lab portals

- + Latest models, minimal ops
- Rate limits, policy shifts, data control concerns

Examples: OpenAI, Perplexity, Gemini, OpenRouter

Hyperscalers (managed)

- + Managed scaling and tooling
- Cost surprises, limited customization, provider lock-in

Examples: Azure AI, Vertex AI, SageMaker

Private/Sovereign AI

- + Data control, cost governance, custom releases
- You own hardware, serving, and reliability

Runtimes: llama.cpp, vLLM, TGI; Platforms: K8s, bare metal

Private and Sovereign AI

Own your stack from weights → runtime → platform.

Infra

Linux, GPUs, Kubernetes, observability

Serving

llama.cpp (CPU/GGUF) & wrappers,
vLLM (GPU), TGI

Governance

Security, access, model registries,
evaluations

Transformers

What it is

Hugging Face Transformers: the de-facto Python library for loading, tokenizing, and running foundation models with consistent APIs.

Strengths

- • Huge model hub, consistent pipelines, rich tokenizers
- • Integration with PEFT, bitsandbytes, evals, datasets
- • Easy prototyping; broad community & docs

Trade-offs

- • Not a high-throughput server by itself
- • Requires pairing with runtimes (vLLM/TGI) for prod serving

Example

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

model_id = "meta-llama/Llama-3-8B-Instruct"
tok = AutoTokenizer.from_pretrained(model_id)
mdl = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=torch.bfloat16)

prompt = "Why private AI?"
ids = tok(prompt, return_tensors="pt").to(mdl.device)
out = mdl.generate(**ids, max_new_tokens=64)
print(tok.decode(out[0], skip_special_tokens=True))
```

Pair with vLLM/TGI for production serving and throughput.

llama.cpp

What it is

High-performance C/C++ inference for LLaMA-family and many HF models via GGUF. Great for CPU, small GPUs, and edge.

Strengths

- • No heavyweight Python/driver stack; portable binaries
- • Runs offline; easy to embed
- • Broad quantization support (Q4_0...Q8, K-quant)

Trade-offs

- • Max throughput lower than GPU servers for big models
- • Use GGUF conversions; some features vary by model

Local demo

```
# Download GGUF (example)
curl -L -o llama3.Q4_K_M.gguf https://huggingface.co/

# Run with llama.cpp
./main -m llama3.Q4_K_M.gguf -n 128 -p "Why private AI?"
```

Tip: Use GPU backends (Metal/CUDA/Vulkan) when available for speedups.

vLLM

What it is

Open-source LLM serving (UC Berkeley).

High-throughput, low-latency GPU inference with PagedAttention and efficient KV cache.

Strengths

- • Excellent throughput with batching/continuous batching
- • OpenAI-compatible server; easy clients
- • Works with HF models; supports tensor/quant formats

Trade-offs

- • Wants decent GPUs and VRAM; plan for scheduling
- • Operational tuning (batching, max_tokens, tensor parallel)

OpenAI-compatible server

```
# Start vLLM server
vllm serve meta-llama/Llama-3-8B-Instruct \
  --host 0.0.0.0 --port 8000 \
  --gpu-memory-utilization 0.9 \
  --max-model-len 8192

# Query (OpenAI API)
curl http://localhost:8000/v1/chat/completions \
  -H 'Content-Type: application/json' \
  -d '{
    "model": "meta-llama/Llama-3-8B-Instruct",
    "messages": [{ "role": "user", "content": "Give me 3 !
  }'
```

Tunables: tensor/pp, max_tokens, trust_remote_code, quantized weights.

What if we cut costs in half?

Quantization shrinks weights and KV cache → fewer/lower-VRAM GPUs, faster tokens/sec.

Approaches

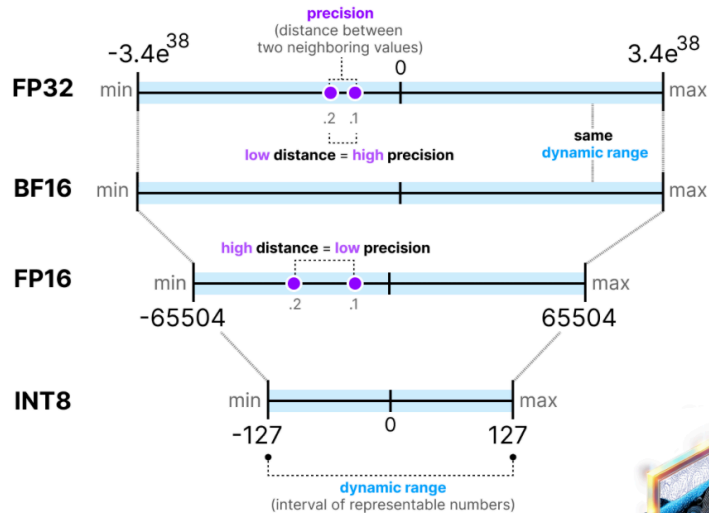
AWQ, GPTQ, GQA-aware, INT8/INT4, FP8

Typical wins

2× VRAM reduction, similar quality with right calibration

Trade-offs

Slight perplexity/accuracy hit; layer-wise sensitivity matters



Quantization demo

Plan

Start with FP16 Llama-3.1 70B Instruct, apply FP8 activation quant with llm-compressor, then serve with vLLM.

Steps

1. Download model weights (HF)
2. Quantize with llm-compressor (FP8 activations on Linear)
3. Launch vLLM with compressed checkpoint
4. Compare latency and VRAM

Example commands (llm-compressor)

```
# Install
pip install llmcompressor transformers

# Python: FP8 dynamic quantization with offload on 2 GPUs
python - <<'PY'
from transformers import AutoModelForCausalLM
from llmcompressor.modifiers.quantization import QuantizationModifier
from llmcompressor.transformers.compression.helpers import get_model_name
from llmcompressor import oneshot

model_stub = "meta-llama/Llama-3.1-70B-Instruct"
device_map = calculate_offload_device_map(
    model_stub,
    reserve_for_hessians=False,
    num_gpus=2,
    torch_dtype="auto"
)
model = AutoModelForCausalLM.from_pretrained(
    model_stub,
    device_map=device_map,
    torch_dtype="auto"
```

Local → Production

Local

- llama.cpp binary, GGUF weights
- vLLM single-GPU serve for quick iteration
- Experiment in notebooks/Transformers before serving

Production

- Kubernetes + GPU scheduling (time-slicing/MIG)
- Autoscaling, observability, circuit-breakers, canaries

SRE checklist: health probes, request budgets, token limits, logging redaction, evals, red-team prompts, rollout policy.

Takeaways

Open models are thriving
across sizes and modalities

Private + sovereign AI gives
control, privacy, and cost
governance

Serve locally with llama.cpp
(CPU/edge) or vLLM/TGI
(GPU scale)

Quantization can halve GPU
needs with minimal quality loss

**You might still have 99
problems – LLMs ain't
one**

Thanks!

Resources

- [Quantization accuracy at scale \(Red Hat article\)](#)
- [LLM Compressor \(quantization library\)](#)
- [vLLM \(OpenAI-compatible server\)](#)
- [Red Hat AI models on Hugging Face](#)
- [ramalama \(containers / local LLM tooling\)](#)
- [llama.cpp \(GGUF inference\)](#)

Follow

@cedricclyburn · YouTube, X/Twitter, GitHub, LinkedIn



LinkedIn



Slides