NOOR MKDAD

Backend and Platform Engineer | AI Inference, Data Systems and Reliability

Email: noorhayatywadam@gmail.com • Location: Orange County, CA (Remote, US) • Work Authorization: U.S. Citizen, no sponsorship
LinkedIn: linkedin.com/in/noor-mkdad-6b0a77293 • GitHub: https://github.com/StanzaAPI • OSS: https://github.com/DrMkdaddy • Live Platform: https://stanzaapi.com

SUMMARY

Backend and platform engineer who builds and operates the systems he ships, and the parts where being wrong is expensive: ledgers, payment paths, cost ceilings and the gates that stop a bad request. 12 production APIs on Cloudflare Workers, a self-hosted inference stack priced at 19 to 41 cents per million tokens, and eleven public repositories where every benchmark below reproduces from a committed harness.

PROFESSIONAL EXPERIENCE

Stanza API — Platform and Systems Engineer

Remote (US) · January 2025 – Present · stanzaapi.com

- **Built and operate** 12 production APIs for healthcare claims, payments and supply-chain data on Cloudflare Workers, behind isolate service bindings and fail-closed API key verification, each with an OpenAPI 3.1 contract.
- **Wrote** a zero-regex ANSI X12 streaming parser that validates 255,000 claim records in 14.2 seconds inside a 25 MB V8 old-space cap, keyed so the consumer walks 255,001 frames instead of 20,655,000 raw segments; the public benchmark harness reproduces it against two competing parsers, including where it loses.
- **Built** a compiler that emits typed clients in TypeScript, Python, C# and Rust: 44 packages published to npm, PyPI, NuGet and crates.io from one spec per API.
- **Shipped** the official MCP server exposing all 12 APIs as agent-callable tools over stdio and HTTP/SSE, and enforced the monorepo merge gate: zero-mock coverage plus linters that catch tests written to pass.

Independent Engineering Practice — Platform and Inference

Orange County, California · June 2024 – Present · github.com/DrMkdaddy

- **Built** an OpenAI-compatible inference gateway on one consumer GPU and **optimized** a llama.cpp Vulkan kernel for RDNA4, both public with their benchmark harnesses; the kernel patch is open upstream (ggml-org/llama.cpp#31022) and the gateway prices every model on the card at 19 to 41 cents per million tokens.
- **Wrote** the declarative infrastructure the work runs on: OpenTofu provisioning and a flake-based multi-host NixOS configuration with a runbook per failure mode, so the lab rebuilds from source rather than from memory.

PUBLIC PROJECTS

Inference and GPU serving

- **local-inference-stack:** an OpenAI-compatible gateway for one 16 GB AMD RX 9070 XT, with a Prometheus endpoint, an MCP server so agents can call the local model as a tool, and a Nix flake that pins the environment. 101 tok/s generation, 2,400 tok/s prefill and 286 tok/s across four concurrent slots at 9.9 GB VRAM. The routing sends each request to the cheapest model that clears a quality tier and a per-request ceiling, refuses rather than overspend, and accounts spend per model on `/cost` and in Prometheus.
- **llama-vulkan-matvec-rdna4:** a performance change to llama.cpp's Vulkan `mul_mat_vecq` path for K-quants - hoisting dequantisation out of the token-column loop, widening output-row blocking from 4 to 8, and dropping the 4x k-unroll at 8 rows to stop register spills. 437 to 495 tok/s aggregate at 8 slots, 14.75 to 12.56 ms per step, single-stream unchanged at 117 tok/s because it is a batching win, and 8/8 greedy outputs bit-identical. Two earlier revisions went up as #31021 and #31024 and were closed unmerged; #31022 is the open one.
- **kv-serving-scheduler** (Go): a block KV allocator, admission control, priority and WFQ queues, and recompute preemption in front of llama.cpp. The real proxy completed 79 of 79 requests at 4 req/s; under mixed priority the high-priority tenant held a 74 ms TTFT p99 while the low-priority tenant absorbed 12.2 s and 14 preemptions, and a calibrated simulator compares policies without burning GPU hours.
- **paged-attention-lab:** a block-paged KV cache with a per-sequence block table, bit-exact against contiguous attention (0.0 maximum absolute error), admitting 54 sequences where 32 fit the same 4096-block budget - and the 5.5x to 25.8x read indirection it costs, measured rather than hidden, plus 23.6% to 58.2% of slot waste avoided.
- **speculative-decoding-lab:** an 8B target at 1.129x with 40/40 byte-identical outputs, 64.7% acceptance against a computed 50.9% break-even. Longer draft lengths lose exactness (n4 is 1.081x with 26/40) and the same trick is 0.875x on a 4B target, so the boundary is published rather than the one good number.

- **quant-quality-harness:** perplexity and KL divergence against an F16 reference for five GGUF levels with speed and VRAM from one config - Q4_K_M costs 4.2% perplexity and 0.0285 KL for 201 tok/s in 2.17 GiB, against Q8_0 at 0.3% for 138 tok/s in 3.41 GiB, so the quant choice is a priced tradeoff instead of a default.
- **valkey-kv-l2:** an L2 for the KV cache with content-addressed blocks, restoring 200 of 200 sequences where a fixed pool served 4, at 1.7 ms mean restore and 60 MB of memory. Priced against prefill on the same machine, it only wins past roughly 4,000 prefix tokens (1.03x at 4,096, 0.90x at 2,048), which is where it switches on.
- **qwen-moe-stream:** a dense 4B GGUF repartitioned into an 8-expert MoE that llama.cpp loads, reproducing the dense model (PPL 17.6157 against 17.6203, 96.47% identical top token), with a CPU executor that streams experts from disk to run a 6.25 GB model in 0.775 GB of RSS. The lossy router frontier is measured: top-6 reads 75% of FFN bytes at 1.186x dense perplexity, top-7 reads 87.5% at 1.011x.
- **llama-backend-compare:** settled HIP against Vulkan on gfx1201 instead of assuming - ROCm 7.2.4 was intermittent, one clean session then a hang at model load, so Vulkan became the serving path. The failed experiment is committed as a configuration study with runbooks, not as the one lucky run.

Data and test corpora

- **hf-test-datasets** (huggingface.co/StanzaAPI): 14 CC0 test-data datasets covering IBAN, ISO 20022, X12, Peppol and EPCIS, with cards, citations and computed check digits rather than random strings - the fixtures the parser and validator suites run against.

Infrastructure

- **platform-infra:** OpenTofu provisioning for the Proxmox host with the Proxmox provider, a runbook per failure mode, and a public history that shows the rebuild path rather than a screenshot of a finished lab.
- **NixOS-Config:** a flake-based multi-host NixOS configuration covering the hosts, the services on them and the users who reach them, so a host is defined by code that can be diffed and rolled back.

OPEN SOURCE

Contributed 37 pull requests across 18 upstream repositories, eight merged - two in rerun-io/rerun (11.6k stars), three in public-apis, and a merged Flux scheduler fix in stable-diffusion.cpp - with an open kernel optimization in llama.cpp (#31022) and an open robustness fix in the MCP TypeScript SDK.
Published the 14 CC0 test-data datasets above under the StanzaAPI organisation on Hugging Face, kept current with the parser suites that consume them.

TECHNICAL SKILLS

Languages: Go, TypeScript, Python, Rust, SQL, Bash
Platform: Cloudflare Workers (D1, KV, R2, Vectorize, Workers AI), PostgreSQL, SQLite, SSE, NDJSON, OpenAPI 3.1, MCP, ANSI X12, ISO 20022, Peppol
Infrastructure: Linux (Debian, Ubuntu), Proxmox VE, LXC, KVM, ZFS, NixOS flakes, OpenTofu, Kubernetes, Helm, Nginx, WireGuard, Prometheus, Grafana, Git, GitHub Actions, Authentik SSO, OIDC/JWT
Inference: llama.cpp, Vulkan kernels, KV cache, paged attention, speculative decoding, GGUF quantization, OpenAI-compatible gateways, cost-aware routing