我们做了一些尝试

第一次尝试 sglang 与 vLLM 部署大模型的心路历程

Atlas 800I A2 ·4 节点 × 8 卡 Ascend 910B2

2026-07

目录

序章：一套崭新的昇腾集群

我们手上有一套 **Atlas 800I A2** 集群：4 个节点，每节点 8 张 **Ascend 910B2**（64 GB HBM2e），一共 32 张卡。节点内是 HCCS 全互连，节点之间靠 NPU 板载的 **8 × 200G RoCE** 光口互通，MoE 的跨机 all-to-all 就压在这条链路上。CPU 是鲲鹏-920，每节点 2 TB 内存、7 TB NVMe。软件栈 CANN 9.0.0，固件 7.7.0.9.220。

目标很朴素：把当下最强的几个开源大模型（GLM-5.1、GLM-5.2、DeepSeek-V4-Pro、MiniMax-M2.7）在这套国产卡上跑起来，跑稳、跑快、能对外服务。

这是我们第一次在昇腾上认真部署大模型。回头看，几乎每一个模型都逼我们踩穿一层新的坑，也逼我们对”昇腾能做什么、不能做什么”有了越来越清晰的认知。这份文档把这段路完整记下来——不只是最终能用的配置，更是为什么是这套配置。

下面按我们实际走过的顺序，一个模型一章。

第一章：GLM-5.1 —— 第一个模型，最深的坑

第一个模型最难，因为所有基础设施的坑都要在它身上踩一遍。

坑一：驱动装不上，只有一个错误码 0x0091

装昇腾 HDK 驱动 `.run` 包，直接返回 `0x0091`，没有堆栈、没有提示。查了很久才发现根因：昇腾驱动要求 `HwHiAiUser` 用户和用户组必须预先存在，安装程序要把设备权限挂到这个用户上，用户不在就直接拒绝。

```
groupadd HwHiAiUser
useradd -g HwHiAiUser -d /home/HwHiAiUser HwHiAiUser
./Ascend-hdk-910b-npu-driver.run --full --install-for-all
```

坑二：RoCE 网络，两个隐蔽的大坑

第一个反直觉的地方：`HCCL_IF_IP` 要填 **bond0** 管理网的 IP（控制面握手用），而不是 RoCE 网段的 IP。直觉上”数据走 RoCE 就该填 RoCE IP”，填错的结果是 Gloo 退回去连 `127.0.0.1`，节点间握手永久失败。真正的数据面（MoE EP all-to-all）由 910B 自动走板载 RoCE 口，根本不用手工指定。

第二个坑：节点防火墙默认只放行 22 端口，HCCL 握手端口和 RoCE 流量被默默拦掉，现象是握手超时。必须放行内网两段：

```
iptables -I INPUT 1 -s 192.168.5.0/24 -j ACCEPT   # 管理网
iptables -I INPUT 1 -s 100.97.0.0/24 -j ACCEPT    # 板载 RoCE
```

坑三：SGLang EP —— 一条彻底的死路

社区里 SGLang 的专家并行（EP）口碑不错，我们优先尝试。启动 EP 立刻报错：

```
aclnnDispatchFFNCombine not in libopapi.so
```

翻译过来：**SGLang EP** 依赖的融合算子，在昇腾算子库里根本没有实现。我们换 CANN 版本重试，报错纹丝不动——这不是版本问题，是算子从未落地。SGLang EP 在 910B 上是死路。

结论下得很快：换框架，转向 **vLLM-Ascend v0.19.1rc1**，它原生支持 external DP + EP，路通了。

坑四：FP8 权重，硬件不认

GLM-5.1 官方放出的是 FP8（NVIDIA E4M3 格式）权重。问题是：**910B2** 的 **Cube** 计算单元只认 **FP16 / BF16 / INT8**，没有 **FP8** 数据通路。这是硬件边界，软件绕不过去（要到 910D 才支持 FP8）。

我们把几种格式都摸了一遍：

权重格式	能否在 910B 跑	结论
FP8 (E4M3)	硬件不支持	直接放弃

权重格式	能否在 910B 跑	结论
BF16	可以	精度好但显存紧
W8A8 (INT8)	可以	速度/显存最均衡，生产选它
W4A8 (INT4)	能跑	省 45% 体积，但实测慢 38–57%

最终用 msModelSlim 产出的 **W8A8** 权重，`--quantization ascend` 加载。

坑五：多机握手，苦等 30 分钟然后崩

vLLM external DP 启动时，rank0 要等齐 4 个节点。有一次用脚本批量拉起，一台因为 SSH 瞬时抖动漏启了，**rank0** 苦等 **30** 分钟（`3/4 clients joined`），最后超时，整个集群崩掉。从此定了规矩：逐节点确认起来，重启前先 `docker restart vllm` 释放残留显存。

最终跑通的配置

external DP=4 × TP=8 + EP，32 卡。各节点只有 `--data-parallel-rank` 和 `HCCL_IF_IP` 不同：

```
export HCCL_IF_IP=192.168.5.<本节点bond0>
export GLOO_SOCKET_IFNAME=bond0 TP_SOCKET_IFNAME=bond0 HCCL_SOCKET_IFNAME=bond0
export HCCL_OP_EXPANSION_MODE=AIV HCCL_BUFFSIZE=1024 TASK_QUEUE_ENABLE=1
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

vllm serve /models/GLM-5.1-w8a8 \
  --host 0.0.0.0 --port 8000 \
  --data-parallel-size 4 --data-parallel-rank <0|1|2|3> \
  --data-parallel-address 192.168.5.4 --data-parallel-rpc-port 13389 \
  --tensor-parallel-size 8 --enable-expert-parallel \
  --speculative-config '{"num_speculative_tokens":1,"method":"mtp"}' \
  --quantization ascend \
  --max-num-seqs 256 --max-model-len 202752 --max-num-batched-tokens 4096 \
  --trust-remote-code --gpu-memory-utilization 0.95 --enable-chunked-prefill \
  --served-model-name glm-5.1
```

性能与最重要的一课

版本	关键改动	结果
v1 基线	无 MTP	输出 ~1040 tok/s
v3	叠加 MTP 投机解码	吞吐 **+41~73%**，TPOT −40%
v5	batched 2048→4096	无缓存 RPM 204→340（+67%）

真正的洞察不在参数，而在缓存：

- 对话场景，缓存命中 0% vs 83%，RPM 差 **2.97×**；
- RAG 场景，0% vs 91% 命中，RPM 差 **5.4×**。

我们早期那些” 漂亮的 RPM 数字”，全是拿固定 prompt 打出来的高命中值，偏乐观。缓存命中率是这套系统最大的杠杆（**3–5** 倍），比任何单个参数都重要。而 **MTP** 投机解码是性价比最高的一步：一行参数，吞吐涨 40–70%，质量无损。

第二章：MiniMax-M2.7 —— 有了地基，也有新坑

有了 GLM-5.1 的地基，MiniMax 顺很多。它是 230B 总参、只有 **10B** 激活的 MoE，算力需求低，我们直接用 **BF16** 跑，不量化——32 卡每卡才 ~15 GB 权重，显存绰绰有余。而且它是标准 softmax + GQA 注意力，不碰昇腾的稀疏算子墙。

但还是踩了三个新坑，且每个报错都在”指错方向”。

坑一：模型没分发，报了个”格式错误”

三台 worker 一启动就崩，报 `OSError: Repo id must be in the form 'namespace/repo_name'`。看着像模型名格式错了，实则是：模型只下到了主节点，**worker** 本地没有，vLLM 把不存在的本地路径当成 HF repo id 去解析，抛出这个误导性错误。external DP 每个节点都必须有本地副本——先 rsync 分发（内网 ~280 MB/s，1.28 TB 约 1 小时）。

坑二：顺手抄来的一个 **env**，成了新坑

我们照搬 GLM 配置时，顺手带上了 `VLLM_ASCEND_ENABLE_FUSED_MC2=1`。结果加载握手都过了，一到真正推理就崩：

```
aclnnMoeDistributeDispatchV4 failed code 561000
HcclAllocComResourceByTiling ret=4
```

根因：MiniMax 的 MC2 融合 MoE 通信算子在本环境无法分配 HCCL 资源。去掉这个 env（连带 `FLASHCOMM1`），退回标准 all2all，就好了。教训：从别的模型移植 env 时，每一个”优化项”都要重新审视，它可能是别人的解药、你的毒药。

坑三：压测全 **404**

压测 0 成功，全是 `404`。原因很蠢也很典型：bench 脚本默认 `--model glm-5.1`，而新服务名是 `minimax-m2.7`，vLLM 对不匹配的 model 名返回 404。报错方向（服务问题）和真实问题（客户端参数）完全对不上。

最终配置与性能

配置和 GLM-5.1 几乎一样，去掉 `--quantization`，换 MiniMax 的 reasoning / tool 解析器：

```
vllm serve /models/MiniMax-M2.7-bf16 \
  --data-parallel-size 4 --data-parallel-rank <0|1|2|3> \
  --data-parallel-address 192.168.5.4 --data-parallel-rpc-port 13389 \
  --tensor-parallel-size 8 --enable-expert-parallel \
  --max-num-seqs 64 --max-model-len 196608 --max-num-batched-tokens 2048 \
  --trust-remote-code --gpu-memory-utilization 0.90 --enable-chunked-prefill \
  --reasoning-parser minimax_m2_append_think --tool-call-parser minimax_m2 \
  --served-model-name minimax-m2.7
```

场景	并发	RPM	输出 tok/s	TPOT
常规 2048/256	128	446	1903	66 ms

场景	并发	RPM	输出 tok/s	TPOT
RPM 峰值	512	**1753**	**3740**	67 ms

MiniMax 快得惊人：RPM 2–3× 于 GLM，TPOT 快 2–2.8×（10B 激活 vs ~40B）。但我们做了一个反直觉的商业发现：按官方 3 折卖 token，**GLM** 反而每卡每天赚更多——MiniMax 能扛 3× 流量，但单 token 官方价便宜 4×，折算下来 GLM 收益高 1.4–2×。性能第一，未必是收益第一。

第三章：GLM-5.2 —— 稀疏注意力的生态债

GLM-5.2 换了架构：**DSA** 稀疏注意力（每层 **index_topk=2048**）+ **MTP**，对应 sglang 里的 `GlmMoeDsaForCausalLM`。这一章的主题是”新架构的算子生态还没长齐，我们替它补窟窿”。

坑一：vLLM 直接不认 → 只能 sglang

vLLM-Ascend 缺 DSA indexer 算子，加载 `GlmMoeDsaForCausalLM` 直接报错退出。**GLM-5.2** 只能走 **sglang**，这是整段旅程的起点，也没得选。

坑二：sglang 无条件构造 Indexer → 运行时打补丁

启动即崩，`NoneType has no create_weights @ layer3`。根因：sglang 对全部 78 层无条件构造 DSA Indexer，但 GLM-5.2 只有 22 个 full-attention 层带 indexer 权重，其余 shared/skip 层没有 `wq_b`，一构造就崩。

我们写了 `patch_glm52_indexer.py`，运行时打补丁——只对 full 层构造 Indexer，其余跳过。这个补丁从此成了每次 **serve** 前的强制前置步骤（幂等安全）。

坑三：deepep 跨节点挂死 → 退回 moe-none

启用 `--moe-a2a-backend deepep`（normal / auto 模式），首次跨节点 MoE dispatch 就挂死：

```
aicore timeout 507014 / NotifyDispatchA2 tiling fail
```

这是 A2 上 deepep 跨节点 all-to-all 的算子层 bug，绕不过。只能改成 `--moe-a2a-backend none`，让 MoE 退化成纯 TP32 all-reduce——用通信量换稳定，代价是放弃 deepep 潜在的 decode 收益。（这个坑，后面 **W4A8** 那一章会以更凶的形式再次出现。）

坑四：eager 慢到 2 tok/s → 图模式 8× 起飞

第一次推理慢得离谱，decode 只有 ~2 tok/s。根因是 sglang 默认 eager 模式，每步实时编译 NPU 算子，overhead 巨大。启用 `--cuda-graph-backend-decode full` 捕获解码图后，吞吐飙到 ~16 tok/s（约 **8×**）。

但图模式带来一条铁律：运行批一旦超过 `--cuda-graph-max-bs`，**decode** 会掉回 **eager**，**TPOT** 从 **82ms** 暴涨到 **700+ms**。所以 `cuda-graph-max-bs` 必须不低于实际 P95 运行批。

坑五：MTP 在 910B2 上是净亏损（试了 11 次才定论）

GLM-5.2 自带 MTP 权重，理论上 decode 能 +60%。但在 910B2 上，draft 图捕获封顶 bs=32，加上 verify forward 的代价，图模式下 **NEXTN** 反而慢 **2.5×**（93 tok/s vs 230 tok/s）。同样一个特性，在 H100 上是 +1.5×，在 910B2 上是负收益。生产不用 **MTP**——这和 GLM-5.1（vLLM 路线 MTP 大赚）恰好相反，框架和硬件的组合决定一切。

最终配置与性能

`moe-none + 图模式 + Tier-1 融合 + HiCache`，在不需要任何新算子的前提下把 co-located TP32 榨到极限：

```
export HCCL_IF_IP=<本机bond1>
export GLOO_SOCKET_IFNAME=bond1 TP_SOCKET_IFNAME=bond1 HCCL_SOCKET_IFNAME=bond1
export HCCL_BUFFSIZE=512 HCCL_OP_EXPANSION_MODE=AIV TASK_QUEUE_ENABLE=1
export USE_PA_DECODE=1 USE_PA_PREFILL=1 ASCEND_USE_FIA=1
export HCCL_INTRA_PCIE_ENABLE=1 HCCL_INTRA_ROCE_ENABLE=0
export SGLANG_NPU_USE_MLAPO=1 SGLANG_USE_FIA_NZ=1 SGLANG_NPU_FUSED_MOE_MODE=2
# 注意：切勿设 SGLANG_NPU_USE_MULTI_STREAM（图捕获冲突 → 107025 崩）

python3 /sglog/patch_glm52_indexer.py   # 必须，DSA 稀疏层补丁

python3 -m sglang.launch_server --model-path /models/GLM-5.2-w8a8 \
  --quantization modelslim --page-size 128 \
  --nnodes 4 --node-rank <RANK> --dist-init-addr 192.168.0.72:30000 --tp-size 32 \
  --attention-backend ascend --device npu --dtype bfloat16 \
  --mem-fraction-static 0.72 --context-length 1048576 \
  --chunked-prefill-size 8192 --max-running-requests 512 \
  --moe-a2a-backend none --enable-hierarchical-cache \
  --cuda-graph-backend-decode full --cuda-graph-backend-prefill disabled \
  --cuda-graph-max-bs 128 --disable-overlap-schedule --trust-remote-code \
  --tool-call-parser glm45 --reasoning-parser glm45 \
  --host 0.0.0.0 --port 10010 --served-model-name glm52
```

场景	并发	RPM	prefill tok/s	TPOT
稳态 in256/out16	384	1577	—	82 ms
峰值 in2048/out16	512	**1862.8**	**52,530**	663 ms

我们还顺手把 **PD** 分离（**2P+2D**）跑通了 1M context——KV 传输走华为 MemFabric（`--disaggregation-transfer -backend ascend`，不是 Mooncake）。但 32 卡规模 PD 分离不划算（RPM 仅 co-located 的 1/16，权重要载两份），定性为实验留档：**PD** 分离的收益要不低于 **96** 卡才谈得上。

第四章：DeepSeek-V4-Pro —— env 会被偷偷吃掉

DeepSeek-V4-Pro 是最硬的骨头：Compress-4-Attention（非纯 MLA），W4A8 + MTP。我们走了两条路——vLLM-Ascend（W4A8 预量化）和 sglang（自建 W8A8）。

坑一：HCCL TLS 跨节点不一致，专家组必崩

建 MC2 专家组时报 `error 1 / EI0016 tls invalid`。根因极其隐蔽：某个节点的部分卡 **TLS** 开着，其余卡/节点关着，状态不统一，MC2 组就建不起来。修复要在每节点、每张卡上执行，一张都不能漏：

```
for i in 0 1 2 3 4 5 6 7; do hccn_tool -i $i -tls -s enable 0; done
hccn_tool -i 0 -tls -g   # 验证全部 switch[0]
```

坑二：--enforce-eager 之祸，TPOT 646ms

为了先”跑起来”我们加了 `--enforce-eager`，确实跑通了，但慢到无法接受——**TPOT 646ms**。定位是图编译被禁，61 层逐 op 同步派发。去掉它、改 `FULL_DECODE_ONLY` 图模式，**TPOT** 立刻降到 **108ms**，快了 **6×**（MTP draft 头单独保留 enforce_eager 即可）。

坑三：vLLM v1 DP 会剥离环境变量

最诡异的一个坑：跨节点 gloo 初始化失败，回环到 `127.0.0.1`。根因是 **vLLM v1** 的 **DP engine-core** 在 **spawn** 子进程时，剥掉了几乎所有环境变量，只留 `HCCL_IF_IP`，于是 `GLOO_SOCKET_IFNAME` 这些全丢了。解法很硬核——写一个 `sitecustomize.py` 装进 site-packages，让每个 **python** 进程一启动就自动重注入这些 ifname：

```
# sitecustomize.py (装入 site-packages)
import os
for k in ("GLOO_SOCKET_IFNAME","TP_SOCKET_IFNAME","HCCL_SOCKET_IFNAME"):
    os.environ.setdefault(k, "bond1")
```

坑四（sglang 路线）：FP4 不可行，自己编算子

FP8 原始权重里 `expert_dtype=fp4`，而昇腾没有原生 **FP4** 算子、也没有在线反量化，FP4 路径是 CUDA 专用。必须量化成 W8A8 才能上 NPU。

更狠的是：sglang 的 `dsv4` 注意力后端要调 6 个自定义算子，社区旧镜像是 V4-Flash 原型、算子写死 **d=4096**，而 V4-Pro 是 **d=7168**，直接用会静默出错。我们只能用 AscendC 工具链从源码自己编这套 d=7168 的算子（来自 `hicann/ops-transformer` 和 `cann-recipes-infer`），编成 torch wheel、commit 进镜像。还有个隐藏坑：`set_env` 会把 `ASCEND_CUSTOM_OPP_PATH` 覆盖成只含自己，漏掉 AICPU 算子，得手动把两个 vendor 路径都挂上。

坑五：显存卡死，要等 18 秒

`pkill` 掉 serve 后，残留 worker 卡死、占住 63 GB HBM 不放。重建容器立刻 OOM。规律是：`docker rm -f vllm` 之后要等 ~**18** 秒让驱动回收显存（掉到 ~3.4 GB）再重建。

最终配置与性能（**vLLM W4A8 + MTP**）

```
export HCCL_IF_IP=$LOCAL_IP
export GLOO_SOCKET_IFNAME=bond1 TP_SOCKET_IFNAME=bond1 HCCL_SOCKET_IFNAME=bond1
export HCCL_BUFFSIZE=512 HCCL_OP_EXPANSION_MODE=AIV TASK_QUEUE_ENABLE=1
export VLLM_ASCEND_ENABLE_FLASHCOMM1=1 PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

vllm serve /data/models/DeepSeek-V4-Pro-w4a8-mtp \
  --host 0.0.0.0 --port 10010 $HEADLESS --served-model-name ds-v4 \
  --max-model-len 1048576 --max-num-batched-tokens 2048 --max-num-seqs 512 \
  --gpu-memory-utilization 0.93 --block-size 128 \
  --data-parallel-size 4 --tensor-parallel-size 8 \
  --data-parallel-start-rank $RANK --data-parallel-address 192.168.0.72 \
  --enable-expert-parallel --quantization ascend \
  --model-loader-extra-config '{"enable_multithread_load":"true","num_threads":128}' \
  --speculative-config '{"num_speculative_tokens":1,"method":"mtp","enforce_eager":true}' \
  --compilation-config '{"cudagraph_mode":"FULL_DECODE_ONLY","cudagraph_capture_sizes":[1,8,16,32,64,128]}'
```

指标	值	备注
单流 TPOT	**108 ms**	enforce-eager 时 646ms
最大 RPM	**1056 @ 并发 192**	in256/out16
decode 上限	~115 tok/s	跨机 EP all-to-all 通信受限
加载耗时	~7 min	128 线程并行加载后

decode 顶到 ~115 tok/s 就上不去了，AICore 空等 0–10%——这是跨节点 **EP all-to-all** 的通信硬顶，不是算力不够。这个观察，为下一章 W4A8 的最终结论埋了伏笔。

第五章：GLM-5.2-W4A8 —— 撞穿一堵算子墙

最后一章最曲折。我们想把 GLM-5.2 从 W8A8 进一步压到 **W4A8**（391 GB，省显存、腾出空间做更大 KV）。模型是我们自己量化的。

自建量化：不是免费午餐

官方没有能直接用的 GLM-5.2 W4A8，我们用 msModelSlim 从 BF16 base 自己量化。过程踩了三个坑：缺 `accelerate` 依赖、多卡 **pickle bug**（`exception_handler` 装饰器无法跨进程序列化，用 commit `08422c1` 修）、多卡 **OOM**（碎片化，`expandable_segments:True` 解）。产物 391 GB、102 shard、含 MTP，是标准的混合精度：230,400 层 W4A8 专家 + 1,702 层 W8A8 + 1,420 层 BF16。量化本身很成功。

两个补丁，让 **sglang** 认得这份权重

在 sglang v0.5.14 上跑，需要两个源码补丁：

- **PR #28526**：W4A8 MoE 下 bf16 hidden_states 要先动态量化成 int8（否则 `per_token_scale=[None]` 崩）；
- **share-weight**：DSA shared(skip_topk) 层没有 indexer 权重，要走 `UnquantizedLinearMethod`。

打好补丁 commit 成镜像 `lmsysorg/sglang:v0.5.14-glm52w4a8`，分发四机。单机 **tp8** 完美工作：输出正确、不乱码、图模式快。

撞墙：跨机 **deepep = aicore** 越界，**8** 种配置全崩

真正的墙在跨机。我们想用满 32 卡做一个 200K 上下文的池化实例，于是要跨机 EP。结果无论怎么配，**deepep** 的跨机 **all-to-all** 都在 **NPU** 内核越界崩溃：

```
The DDR address of the MTE instruction is out of range
EE9999: aicore exception
```

我们把能想到的组合全试了一遍：

#	配置	结果
1–2	纯 TP32 + deepep（图 / eager）	aicore MTE 越界
3–6	DP4×TP8+EP（图 / 限批图 / breakable 图 / eager）	aicore MTE 越界
7	任意 + moe-none	能跑，但输出乱码

而唯一能跨机跑的 `moe-none`，对 W4A8 专家的计算路径是乱码（补丁修不了这个）。我们甚至追到 sglang daily build（含一个"A2A 图捕获修复" 的新 commit），结果发现那个修复是给 FlashInfer（CUDA GPU）的，对 NPU 无效；daily build 反而引入了新的 DP 控制器回归。

结论冷峻但清晰：跨机 **deepep**（**EP over RoCE**）在 **910B2 / CANN9** 这套栈上是算子层 **bug**，调参救不了。官方文档里，A2（910B2）也只验证了单机 tp8，跨机 deepep 只在 A3 上验证过。这印证了上一章 DeepSeek-V4-Pro

的观察——跨机 EP 在 A2 上就是不稳。

落地：4 × 单机 tp8 + router

既然单机 deepep 完美、跨机 deepep 死路，那就不跨机：每个节点各跑一个独立的 tp8 副本，前面挂一个 `sglang_router` 做负载均衡。

```
export DEEP_NORMAL_MODE_USE_INT8_QUANT=1
export SGLANG_DEEPEP_NUM_MAX_DISPATCH_TOKENS_PER_RANK=32
export HCCL_INTRA_PCIE_ENABLE=1 HCCL_INTRA_ROCE_ENABLE=0

python3 -m sglang.launch_server --model-path /models/GLM-5.2-w4a8 \
  --attention-backend ascend --device npu \
  --tp-size 8 --nnodes 1 --dp-size 1 --enable-dp-attention \
  --context-length 204800 --mem-fraction-static 0.85 \
  --cuda-graph-max-bs 24 --max-running-requests 24 \
  --enable-hierarchical-cache --hicache-ratio 2 \
  --quantization modelslim \
  --moe-a2a-backend deepep --deepep-mode auto --load-balance-method round_robin \
  --host 0.0.0.0 --port 10010 --api-key sk-glm52 \
  --reasoning-parser glm45 --tool-call-parser glm47
```

router（node0 上跑，入口:8000）：

```
python3 -m sglang_router.launch_router \
  --worker-urls http://192.168.0.72:10010 http://192.168.0.33:10010 \
                http://192.168.0.50:10010 http://192.168.0.38:10010 \
  --host 0.0.0.0 --port 8000 --policy round_robin
```

并发/节点	graph-bs	聚合 tok/s
8	8	344
16	16	619
24	24 + HiCache	**807.5**

峰值 **807.5 tok/s @** 并发 **96**，**96/96** 成功，全程不崩。代价是每个请求被限制在单节点的 KV 池里（~40K 上下文），换不来 200K 单请求——**W4A8 + 200K** 单请求在这套栈上做不到，因为它需要跨机池化 **HBM**，而跨机又必须 **deepep**，**deepep** 又崩。想要 200K，只能回 W8A8（moe-none 纯 TP32 跨机不乱码）。

终章：这一路学到的东西

五个模型走下来，很多坑不是孤立的，而是同一批规律的不同侧面。

1. FP8 / FP4 是硬件边界，不是软件问题。 910B2 只认 FP16 / BF16 / INT8。FP8（GLM-5.1）和 FP4（DeepSeek-V4-Pro）都得先量化成 W8A8 / W4A8 才能上卡。别在软件层跟硬件较劲。

2. 跨机 **deepep** 在 **A2** 上不稳，是反复出现的主题。 GLM-5.2 W8A8 挂死、DeepSeek-V4-Pro decode 通信受限、GLM-5.2 W4A8 aicore 越界——三次都指向同一件事：**A2** 的跨机 **EP all-to-all** 生态没成熟。稳妥的路子是 `moe-none`（纯 TP all-reduce）或干脆不跨机（多副本 + router）。官方也只在 A3 上验证跨机 deepep。

3. 图模式是命门。 eager 慢到不可用（2 tok/s），图模式 8× 起飞。但图有 batch 上限，运行批超了就掉回 eager、TPOT 暴涨。`cuda-graph-max-bs` 必须不低于 P95 运行批，这是每个 sglang 部署的必修课。

4. HCCL 网络有一堆反直觉陷阱。 `HCCL_IF_IP` 填管理网而非 RoCE、防火墙默认挡内网、TLS 状态要每卡一致、vLLM v1 会剥离 env——这些坑单看都小，但任何一个都能让集群”握手超时”然后你查一整天。

5. 缓存命中率是最大的杠杆（**3–5×**），远超任何单参数。所有”漂亮的 RPM”都要问一句：命中率多少？固定 prompt 的乐观值不能当生产指标。

6. MTP 看框架和硬件的组合，不是万灵药。 GLM-5.1（vLLM）上 MTP +41~73%，GLM-5.2（sglang）上 MTP 反而慢 2.5×。同一个特性，换个框架/硬件就从解药变毒药。

7. 性能第一，未必是收益第一。 MiniMax 比 GLM 快 2–3×，但按 token 计价，GLM 每卡每天赚更多。技术指标要落到商业账上看。

8. 报错常常指错方向。模型没分发报”repo 格式错误”、客户端 model 名不对报 404、图编译被禁表现为”慢”而非”错”。学会顺着现象往根因挖，别被第一层报错带偏，是这一路最实用的元技能。

附录：五个部署的性能总表

模型	量化	框架 / 拓扑	峰值吞吐	上下文
GLM-5.1	W8A8+MTP	vLLM · DP4×TP8+EP	1467 tok/s	200K
MiniMax-M2.7	BF16	vLLM · DP4×TP8+EP	RPM 1753 / 3740 tok/s	196K
GLM-5.2	W8A8	sglang ·TP32 moe-none	RPM 1862	1M
DeepSeek-V4-Pro	W4A8+MTP	vLLM · DP4×TP8+EP	RPM 1056	1M
GLM-5.2	W4A8	sglang ·4×tp8 + router	807.5 tok/s	200K*

* W4A8 每请求受单节点 KV 池限制（~40K）；200K 为配置值。

一句话总结：把开源大模型搬上国产昇腾卡，难的从来不是”启动命令怎么写”，而是摸清这块硬件和这套框架的边界在哪里——哪些算子有、哪些没有，哪条路官方验证过、哪条是社区一厢情愿。这份心路历程，记的就是我们一寸一寸摸清这条边界的过程。