

【云端低价训推】 KTransformers+AutoDL+LlamaFactory：随用随租的低成本超大模型「微调+推理」一体化流程

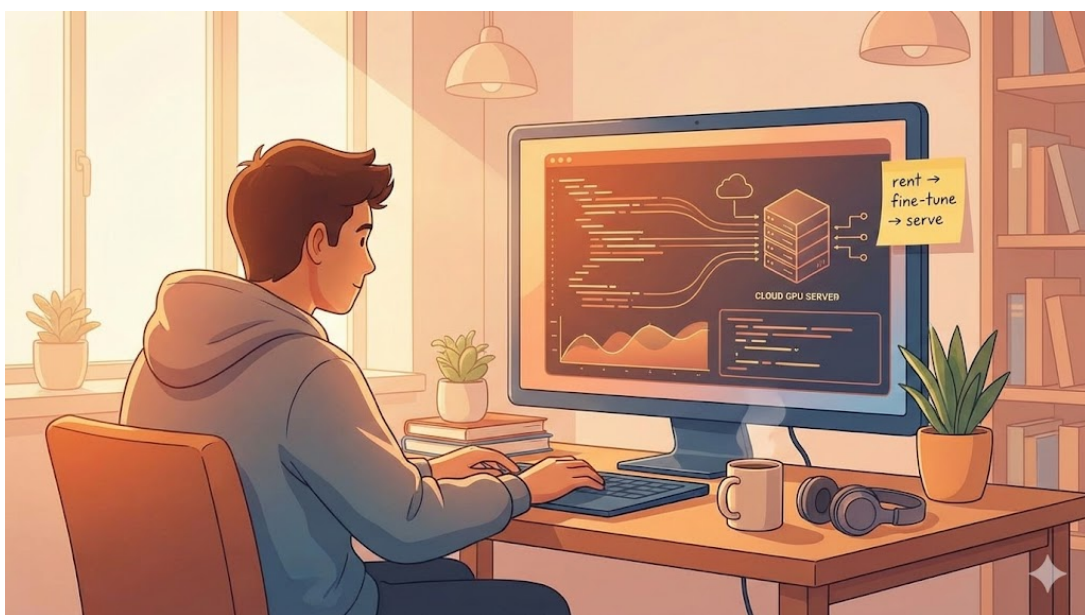
从租机 → 环境 → LlamaFactory--LoRA 微调 → SGLang 上线推理测试 benchmark，一篇用 AutoDL+KTransformers 跑通从 14B~235B 的 Qwen/DeepSeek 模型闭环

Introduction

在大模型研究中，限制实验推进的很多时候不是“有没有想法”，而是“能不能跑起来”：显存门槛高、环境配置复杂、训练—评测—部署链路难以复现，很多好点子停在笔记本里。若你预算有限却想探索大型语言模型（LLM）的微调与推理，本教程会教你如何将 **AutoDL 的随用随租** 与 **KTransformers 的异构加速** 组合成一条“可重复、可扩展、可落地”的实验线路：在相对低的租卡预算下，也能对 Qwen3 / DeepSeek 这类 MoE/大模型完成 **低显存微调（LoRA）**，并将结果以 **稳定的推理进行 benchmark 测试**，便于做评测、写论文、做演示或接入应用。

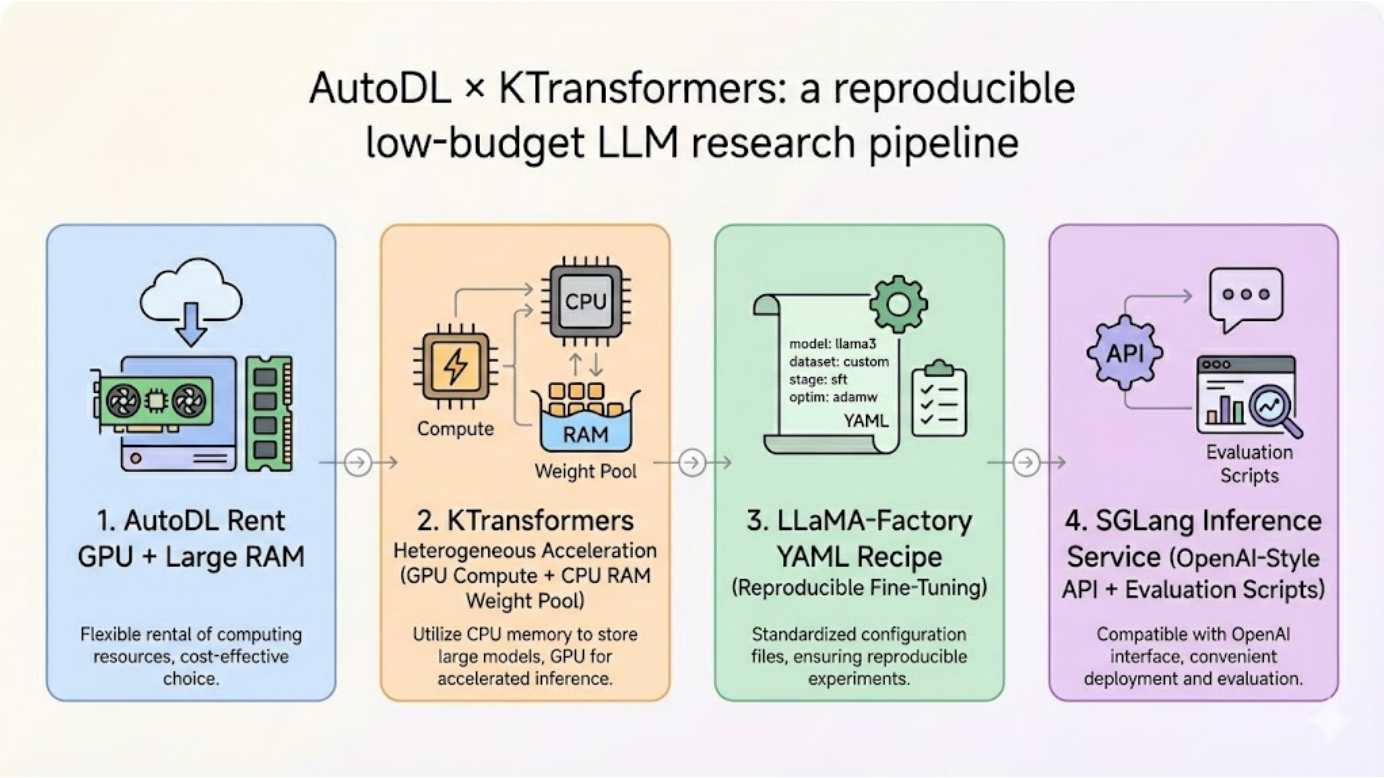
我们在 AutoDL 的实际租用环境中验证过这条路径：**仅需 5 卡 5090（AutoDL 对应 450G 内存）、15 小时、约 215 元**，即可完成 **SOTA 基座模型 Qwen3-235B-A22B 的 LoRA 微调与推理闭环**；或者 **1 卡 5090、2 小时、约 5 元**，实现 Qwen3-30B-A3B 微调的**基础验证**。

价格估算以 5090 为例，具体以下单页面时价为准；数据集选取[风格语气数据集](#)，约 1 万条、408 万 token、3 epoch)



这套方法可以理解为一套“标准化流水线”：把模型、数据与实验设置放进统一流程里，之后换模型/换数据/换实验时，主要替换配置，无需反复重搭工程。实践上，你在 AutoDL 选一台性价比合适的机器并把环境固化为镜像；启用 KTransformers 将“显存门槛”转化为“内存+放置策略”可调的问题；用 LLaMA-Factory 把微调写成可追溯的 YAML 配方；最后用 SGLang 交付统一的推理服务接口，把训练结果直接带入评测与应用。

实验流水线	你需要做什么	你不用再做什么
1. AutoDL 租机器	选一台够用且划算的实例；装好环境并保存为镜像	不必自购高成本服务器；不必在不同机器/实验间反复配环境
2. KTransformers 异构加速	在训练与推理中启用异构能力，支持更大模型探索	不必改底层系统代码；不必被“显存不够”卡死
3. LLaMA-Factory 配方化微调	用 YAML 固化训练参数、数据与产物路径，便于复现与迭代	不必每次手改一堆命令行参数；不必让细节散落在终端历史里
4. SGLang 推理交付	启动推理服务，用统一接口做对话、评测与压测	不必为每次实验重复写推理脚本/服务封装；不必让结果停在本地难以交付



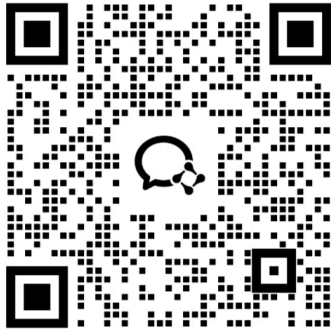
特别案例收集活动（KTransformers-FT Case Study Program）

在大模型研究里，很多时候卡住的不是想法，而是“能不能跑起来”：显存门槛、环境复杂、训练—评测—部署难复现。我们希望把这条 AutoDL × KTransformers × LLaMA-Factory × SGLang 的标准化流水线，沉淀成一批“可复现、可对比、可扩展”的真实案例，让大家在复现论文和做更大规模实验时，少走弯路、能直接复刻起跑。同时，所有想要参与体验的同学也都可以领取一张 **AutoDL** 和**趋境科技**联合发放的 **20 元 5090 代金券**。

我们会在 KTransformers GitHub 新增 Case Studies 板块长期维护：每个合入案例都会生成一张“案例小卡片”（领域/模型规模/资源边界/关键配置/核心结果），便于检索与传播；并定期做社区精选宣传。对特别优质的案例，我们会邀请作者做一次 KT 微调案例分享直播。

如何领取代金券以及提交案例

请添加趋境科技企微账号：（说明来意嗷，大家都可以先领一张来尝试；也欢迎提交案例，能够拿到更多代金券～）



我们想收集什么样的案例

案例的“场景”以研究领域/论文设定为主，尤其欢迎这些方向：

- 1) 论文复现：在公开数据集或可描述的数据构造下复现关键结果，并给出可对比的表格/指标
- 2) Scale-up 扩展：把同一设定扩到更大模型、更长上下文、更低预算，形成清晰的规模化路径
- 3) 微调闭环完整：微调完能直接进入推理评测，形成可复现链路与可交付结果；我们更鼓励不同研究领域的案例（如医疗、法律、金融、教育、科研写作、代码与系统、材料/化学/生物等），让大家能在各自领域里快速找到可参考的配方、数据构造方式与可复现基线。

KTransformers+AutoDL 联动与作者福利

AutoDL 将配合优秀案例的联合推广，包含镜像侧曝光与社群同步宣传。案例合入 Case Studies 后，我们将为作者额外发放 5090 专用代金券；被评为优选案例可获得追加福利。

案例合入后，我们会同步完成三件事：


- 1) 小卡片展示：为案例生成精美小卡片并打标签（领域、模型规模、资源边界、verified/community），便于检索与复刻。
- 2) 社区精选传播：定期汇总优秀案例对外发布，在 GitHub 与社区渠道进行持续曝光。
- 3) 优选案例直播：对表现突出的案例邀请作者进行一次 KT 微调案例直播分享，沉淀可复现经验并扩大社区影响力。

小卡片示例：


KTransformers Case Study


Coding Fine-tuning on Qwen3-30B-A3B


Coding fine-tuning: code generation • bugfix • refactor

 **Model:** Qwen3-30B-A3B


 **Env:** AutoDL • RTX 5090 32GB • RAM 90GB

 **Data:** Coding instructions


 **Method:** LoRA (default)

 **Context:** 8K

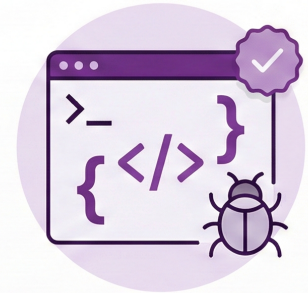
Training → Eval → Deploy

 Fine-tune →  Eval →  Serve

 **HumanEval:** Ready

 **MBPP:** Ready

Result: Adapter trained • Eval loop ready 




GitHub Case
Study Link




KTransformers Case Study


Coding Fine-tuning on Qwen3-30B-A3B

Coding fine-tuning: code generation • bugfix • refactor

 **Model:** Qwen3-30B-A3B

 **Env:** AutoDL • RTX 5090 32GB • RAM 90GB

 **Data:** Coding instructions

 **Method:** LoRA (default)

 **Context:** 8K

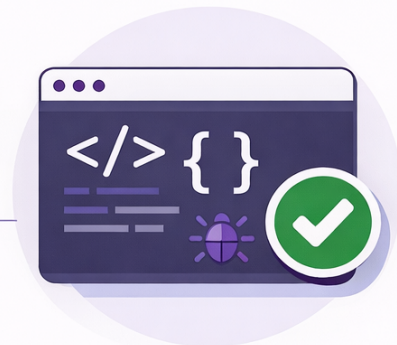
Training → Eval → Deploy

 Fine-tune →  Eval →  Serve

 **HumanEval:** Ready

 **MBPP:** Ready

Result: Adapter trained • Eval loop ready 



QR / Link



GitHub Case Study Link

案例提交模板

字段	说明
标题	领域 + 论文/任务名 + 规模目标（例：医学问答领域指令微调，30B→235B scale-up）
模型与规模	基座模型与参数规模；目标规模与上下文长度（如有）
数据一句话	数据集来源/构造方式与规模（可不公开，最好有统计量）
资源边界	GPU显存 / CPU内存
关键配置入口	LLaMA-Factory 训练 YAML（如有核心修改点）
核心结果	一张小表或截图：效果指标/吞吐时延/成本用时任选其一或组合，能对比就更好
备注	其他想要展示的内容

更进一步：LoRA 变体共创（Heterogeneous Scale-up）

除了收集案例，我们也欢迎**带算法来共创**。如果你正在做新的 **LoRA/Adapter** 变体（更省显存、更快、更稳、或更适配 MoE/长上下文），我们愿意一起把它在异构体系里 scale up：把“论文里的方法”变成“能复现、能扩展、能交付”的工程结果。

你的目标是什么

- 1) 一句话方法：你的改动点是什么，解决哪个瓶颈
- 2) 对齐基线：要对比的标准实现/论文设置
- 3) 验证目标：你最看重的指标（效果/吞吐/显存/成本/稳定性）
- 4) 最小实验：一个可跑的训练入口（代码分支或最小脚本即可）

我们会一起做什么

环节	我们协助的重点
规模扩展	明确资源边界与放大路线（更大模型/更长上下文/更低预算）
异构落地	讨论放置策略与关键算子路径，把瓶颈从“显存不够”转成“内存+策略可调”
复现实验	固化为可追溯配方（训练配置、策略文件、评测脚本），保证别人能一键复刻
社区发布	形成共创小卡片与案例条目，优选方案做分享与传播

共创产出

合入一个可复现的共创案例（含配方与策略入口）+ 一张对比结果表；如果方案成熟，我们会把它纳入 Case Studies 的“共创精选”标签，并优先安排社区分享。

基础使用教程

如果你只想做推理，参考 1→2→5；如果想要 LoRA 微调+推理，参考1→2→3→(4, 可选)→5

1. 租用机器

按照下面三个步骤考虑你对机器的配置需求：

- **先明确目标：只推理，还是要做 KT 微调（LoRA）**
只做推理对 CPU 要求相对低；如果要做 KT 微调，**CPU 必须支持 AMX**。
- **CPU 内存：决定你能否承载大模型权重与中间状态**
模型越大、并发/上下文越高，内存需求越高；这是能不能“跑起来”的关键。
- **GPU 显存：决定你能放多少 gpu experts 来提速（尤其是 MoE）**
gpu experts 越多通常越快，但显存开销也越大；GPU 数量增加也不一定带来线性收益。

Note

KT 微调必须要求 CPU 支持 AMX（Intel 的矩阵加速指令集）。可用：`lscpu | grep -i amx || true` 检查。

推理的最小资源通常≈微调的一半（粗略经验值，受 batch/并发/seq_len/策略影响，仅供起步选型）。

目前 AutoDL 内更推荐使用 **5090 实例**（兼顾推理与微调）。

模型	KT 推理（≈微调 1/2）	KT 微调（参考）
DeepSeek-V2-Lite-14B	3GB 显存 + 15GB 内存	6GB 显存 + 30GB 内存
Qwen3-30B-A3B	3GB 显存 + 30GB 内存	5GB 显存 + 60GB 内存
Qwen3-235B-A22B	9GB 显存 + 225GB 内存	18GB 显存 + 450GB 内存
DeepSeek-V3-671B	35GB 显存 + 0.65TB 内存	70GB 显存 + 1.3TB 内存

以 Qwen3-235B 为例，建议选择 **≥ 450GB 内存** 的实例（例如 **6 × 90GB = 540GB**），并务必确认 **CPU 支持 AMX**。

目前 AutoDL 中，5090 机器的 CPU 多为 **Xeon(R) Platinum 8470Q**（支持 AMX），可同时用于推理和微调。

计费方式:

按量计费

包日

包周

包月

选择地区:

PRO6000

西北B区

北京B区

重庆A区

内蒙B区

北京A区

佛山区

L20专区

V100专区

A800专区

摩尔线程专区

华为昇腾专区

GPU型号:

☐ 全部

☒ RTX 5090 (144/1173)

☐ RTX PRO 6000 (383/1161)

☐ vGPU-32GB (41/1790)

☐ vGPU-48GB (139/520)

☐ H800 (2/104)

☐ RTX 4090D (0/1188)

☐ RTX 4090 (2/1944)

☐ RTX 3090 (0/258)

☐ RTX 3080x2 (0/435)

☐ RTX 3080 Ti (0/380)

☐ RTX A4000 (0/24)

☐ RTX 3060 (0/32)

☐ GTX 1080 Ti (0/14)

☐ CPU (3/334)

☐ CPU-close-HT (0/6)

☐ vGPU-48GB-350W (16/80)

GPU数量:

1

2

3

4

5

6

7

8

10

12

西北B区 / B62机 | cu510g4d4r 可租用至: 2027-02-01

RTX 5090 / 32 GB

空闲/总量 6 / 8

每GPU分配

有 AMX 指令集

硬盘

其它

¥2.39/时

¥3.03/时

7.9折

CPU: 25 核, Xeon(R) Platinum 8470Q

系统盘: 30 GB

GPU驱动: 580.105.08

会员最低享7.9折 ¥2.39/时

内存: 90 GB

数据盘: 50 GB , 可扩容 7980 GB

CUDA版本: ≤ 13.0

6卡可租

租六张卡的情况下, 会有 6*90=540 G 内存

4090 机器的 CPU 多为 Xeon(R) Platinum 8352V (不支持 AMX) , 因此 只适合 KT 推理。

算力市场

AI应用

AI服务器

私有云

帮助文档

更多

计费方式:

按量计费

包日

包周

包月

选择地区:

PRO6000

西北B区

北京B区

重庆A区

内蒙B区

北京A区

佛山区

L20专区

V100专区

A800专区

摩尔线程专区

华为昇腾专区

GPU型号:

☐ 全部

☐ RTX 5090 (1087/3280)

☐ RTX PRO 6000 (0/8)

☐ vGPU-48GB (45/250)

☐ vGPU-48GB-425W (79/160)

☐ RTX 5090 D (3/11)

☒ RTX 4090 (85/984)

☐ CPU (5/44)

GPU数量:

1

2

3

4

5

6

7

8

10

12

北京B区 / 314机 | da2f428b45 可租用至: 2026-05-01

RTX 4090 / 24 GB

空闲/总量 6 / 8

长租特惠

每GPU分配

硬盘

其它

¥2.18/时

¥2.29/时

9.5折

CPU: 16 核, Xeon(R) Gold 6430

系统盘: 30 GB

GPU驱动: 580.76.05

会员最低享9.5折 ¥2.18/时

内存: 120 GB

数据盘: 50 GB , 可扩容 87 GB

CUDA版本: ≤ 13.0

6卡可租

北京B区 / 018机 | 4a2f428b45 可租用至: 2026-05-01

RTX 4090 / 24 GB

空闲/总量 6 / 8

长租特惠

每GPU分配

硬盘

其它

¥2.18/时

¥2.29/时

9.5折

CPU: 16 核, Xeon(R) Platinum 8352V

系统盘: 30 GB

GPU驱动: 580.76.05

会员最低享9.5折 ¥2.18/时

内存: 120 GB

数据盘: 50 GB , 可扩容 340 GB

CUDA版本: ≤ 13.0

6卡可租

2. 环境安装与模型准备

这一节给两种路线：

- 方法一（推荐长期复用）：用 AutoDL 的 miniconda 基础镜像，从源码/whl 安装；装好后可保存为你自己的镜像，后续租机器直接复用。
- 方法二（推荐快速上手）：直接使用 KT 发布的 AutoDL 社区镜像，开箱即用。

环境安装（方法一）：miniconda 镜像 + 源码安装，享受最新版 KT 性能和 feature

Step 1 | 选基础镜像 (miniconda)

在租机器-创建实例的时候，选择 AutoDL 的 miniconda 基础镜像（如下图），把推理与微调所需依赖一次装好，并将环境固化为镜像，后续可直接复用。

计费方式:

按量计费

包日

包周

包月

计费规则

创建完主机后仍然可以转换计费方式。如选择按量计费，价格发生变动以实例开机时的价格为准

选择主机:

主机ID	算力型号/显存	空闲GPU	每GPU分配	CPU型号	硬盘	驱动/CUDA	价格(单卡)
103机	RTX 4090 24GB	5 / 8	CPU: 16核 内存: 120GB	Xeon(R) Platinum 8 352V	数据盘: 50GB 可扩容: 682GB	驱动: 560.35.03 CUDA: 12.6	¥1.98/时 ¥2.08/时

GPU数量:

1

2

3

4

5

6

7

8

数据盘: 免费50GB ☒ 需要扩容

682

 GB

按量计费实例的付费数据盘将按 0.0066元/日/GB在每日24点进行扣款(无论实例是否关机)。使用中可扩容/缩容

实例规格:

GPU型号	CPU	内存	系统盘	数据盘
RTX 4090 * 5卡	80核心	600GB	30GB	免费50GB SSD，付费682GB

镜像:

基础镜像

社区镜像

我的镜像

没有我要的环境?

基础镜像包含常用基本软件，如：深度学习框架、Miniconda等。如需其他软件可创建后安装

Miniconda / conda3 / 3.10(ubuntu22.04) / 11.8

随后即可拿到 ssh 登录方式：

容器实例 实例连续关机15天会释放实例，实例释放会导致数据清空且不可恢复，释放前实例在数据在。

小程序管理实例

租用新实例

批量续费

筛选标签

搜索实例名称/ID

实例ID / 名称	状态	规格详情	本地磁盘	健康状态	付费方式	释放时间/停机时间	SSH登录	快捷工具	操作
内蒙B区 / 103机	运行中	RTX 4090 * 5卡 查看详情	系统盘 0.17% 数据盘 0.00%	正常	按量计费	关机15天后释放 设置定时关机	<div>登录指令 ssh***** 密码 *****</div>	JupyterLab AutoPanel 实例监控 自定义服务	关机 更多

Step 2 | 创建一个 conda 环境并完成安装

还是建议维护两个 conda 环境：在一个环境中同时安装 KTransformers 推理 + SGLang；一个环境中装 KTransformers 微调 + LlamaFactory。下面所有命令在 SSH 连上机器后操作：

```
# 0. Autodl提供了一个学术梯子，可以使用如下
source /etc/network_turbo

# Conda 环境：FOR inference
conda create -n kt-kernel python=3.11
conda activate kt-kernel

# 1. 安装 KTransformers 推理：考虑到暂时kt-kernel存在pip问题，使用下面流程源码安装，后续修复之后可以直接：
pip install kt-kernel-cuda
git clone https://github.com/kvcache-ai/ktransformers.git
cd ktransformers/kt-kernel
```

```

./install.sh

# 2. 安装 SGLang: 请安装 KTransformers 里面指定的这个 SGLang 架构
git clone https://github.com/kvcache-ai/sglang.git
cd sglang
pip install -e "python"

# Conda 环境: FOR SFT
conda create -n kt-sft python=3.11
conda activate kt-sft

# 3. 安装 LLaMA-Factory:
git clone https://github.com/hiyouga/LLaMA-Factory.git
cd LLaMA-Factory
pip install -e .

# 4. 安装 KTransformers 微调: 打两个补丁, 一定不要忽略!
conda install -y -c conda-forge libstdcxx-ng gcc_impl_linux-64
conda install -y -c nvidia/label/cuda-11.8.0 cuda-runtime
# 为了避免本地编译, 推荐直接下载对应版本的 KTransformers 和 flash-attention 的 whl 包, 参考
https://github.com/kvcache-ai/ktransformers/releases/tag/v0.4.2 和 https://github.com/Dao-
AIIab/flash-attention/releases
pip install ktransformers-0.4.2+cu128torch27fancy-cp311-cp311-linux_x86_64.whl
pip install flash_attn-2.8.3+cu12torch2.7cxx11abiTRUE-cp311-cp311-linux_x86_64.whl

```

Step 3 | 保存为镜像 (强烈推荐)

当你确认安装完成后, 建议在 AutoDL 控制台把当前实例保存为你的私有镜像。这样后续你每次租机器, 就不需要重复这整段安装流程。



环境安装 (方法二): KT 定期发布的 AutoDL 社区镜像 (直接可运行), 基础功能快捷使用

如果你的目标是简易使用的话, 可以不从零安装 (方法一), 直接选用 KT 发布的社区镜像, 进入后即可用于 KT 微调与推理 (已预先安装 llamafactory 和 sglang)。

Step 1 | 创建实例时选择社区镜像 (如下图所示路径选择镜像)

使用社区镜像直接克隆进实例

选择主机:

主机ID	算力型号/显存	空闲GPU	每GPU分配	CPU型号	硬盘
268机	RTX 5090 32GB	1 / 8	CPU: 25核 内存: 90GB	Xeon(R) Platinum 8 470Q	数据盘: 50GB 可扩容: 6067GB

GPU数量: 1 2 3 4 5 6 7 8

数据盘: 免费50GB ☐ 需要扩容

实例规格:

GPU型号	CPU	内存	系统盘	数据盘
RTX 5090 * 1卡	25核心	90GB	30GB	免费50GB SSD

镜像: 基础镜像 社区镜像 我的镜像 社区镜像是什么?

暂未选择任何镜像 搜索镜像

请选择镜像

优惠券: 请选择

推荐使用0.5.1的版本，重点修复了之前镜像环境的已知错误

社区镜像

镜像名称

应用名称/ID

ktransformers

搜索结果

deepseek-ai/DeepSeek-R1/deepseekr1_671B_ktra...
全站首发！带你下载、部署满血deepseek_r1 API 非7B...
lccdoge
286

kvcache-ai/ktransformers/KTransformers_offical
KTransformers 微调推理一体化镜像，结合 llamafact...
KTrans...
6

kvcache-ai/ktransformers/ktransformers
Ktransformers极简极速镜像。专门为优化大规模语言...
GPUSnail
50

选择镜像版本

v0.5.1 修复了微调 and 推理的环境不匹配问题
框架: PyTorch:2.9.1 CUDA版本: 12.8 镜像大小: 23.78GB 2026-01-20

v0.4.4 微调推理一体化
框架: PyTorch:2.9.1 CUDA版本: 12.8 镜像大小: 16.68GB 2025-12-30

免费50GB ☐ 需要扩容

规格:

GPU型号	CPU	内存	系统盘	数据盘
RTX 5090 * 1卡	25核心	90GB	30GB	免费50GB SSD

Step 2 | 获取 SSH 登录信息并登录

容器实例 实例连续关机15天会释放实例，实例释放会导致数据清空且不可恢复，释放前实例在数据在。

租用新实例 批量续费

筛选标签 搜索实例名称/ID

实例ID / 名称	状态	规格详情	本地磁盘	健康状态	付费方式	释放时间/停机时间	SSH登录	快捷工具	操作
内蒙B区 / 103机	运行中	RTX 4090 * 5卡 查看详情	系统盘 0.17% 数据盘 0.00% 数据盘 0.00%	正常	按量计费	关机15天后释放 设置定时关机	登录指令 ssh***** 密码 *****	JupyterLab AutoPanel 实例监控 自定义服务	关机 更多

Step 3 | 进入后检查环境是否就绪

如果选择了方法二（社区镜像），进入后通常可以看到相关目录/环境已经准备好（如下图）。此时你只需要打一个补丁如下，然后下载模型并按后文流程运行即可。

-----AutoDL-----

目录说明：

目录	名称	速度	说明
/	系统盘	一般	实例关机数据不会丢失，可存放代码等。会随保存镜像一起保存。
/root/autodl-tmp	数据盘	快	实例关机数据不会丢失，可存放读写IO要求高的数据。但不会随保存镜像一起保存

CPU：25 核心
内存：90 GB
GPU：NVIDIA GeForce RTX 5090, 1
存储：
 系统盘 / : 56% 17G/30G
 数据盘 /root/autodl-tmp: 59% 88G/150G

*注意：
1.系统盘较小请将大的数据存放于数据盘或文件存储中，重置系统时数据盘和文件存储中的数据不受影响
2.清理系统盘请参考：<https://www.autodl.com/docs/qa1/>
3.终端中长期执行命令请使用screen等工具开后台运行，确保程序不受SSH连接中断影响：<https://www.autodl.com/docs/daemon/>
root@autodl-container-e76b459451-dda04498:~# ls
LLaMA-Factory flash_attn-2.8.3+cu12torch2.9cxx11abiTRUE-cp312-cp312-linux_x86_64.whl miniconda3
autodl-pub ktransformers sglang
autodl-tmp ktransformers-0.4.2+cu128torch29fancy-cp312-cp312-linux_x86_64.whl tf-logs
root@autodl-container-e76b459451-dda04498:~# █

小提示：如果你不确定镜像里有哪些 conda 环境，可用 `conda env list` 查看；再按教程要求激活对应环境即可。

模型准备

由于模型文件较大，推荐把模型文件放在 AutoDL 的数据盘（避免系统盘爆掉，如下图 `/root/autodl-tmp`），并用 `huggingface-cli` 下载到指定目录。

* Documentation: <https://help.ubuntu.com>
* Management: <https://landscape.canonical.com>
* Support: <https://ubuntu.com/pro>

This system has been minimized by removing packages and content that are not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Dec 29 14:58:15 2025 from 127.0.0.1

-----AutoDL-----

目录说明：

目录	名称	速度	说明
/	系统盘	一般	实例关机数据不会丢失，可存放代码等。会随保存镜像一起保存。
/root/autodl-tmp	数据盘	快	实例关机数据不会丢失，可存放读写IO要求高的数据。但不会随保存镜像一起保存

CPU：50 核心
内存：180 GB
GPU：NVIDIA GeForce RTX 5090, 2
存储：
 系统盘 / : 41% 13G/30G
 数据盘 /root/autodl-tmp: 38% 57G/150G

*注意：
1.系统盘较小请将大的数据存放于数据盘或文件存储中，重置系统时数据盘和文件存储中的数据不受影响
2.清理系统盘请参考：<https://www.autodl.com/docs/qa1/>
3.终端中长期执行命令请使用screen等工具开后台运行，确保程序不受SSH连接中断影响：<https://www.autodl.com/docs/daemon/>
root@autodl-container-e76b459451-dda04498:~# █

如果你需要从 Hugging Face 下载大模型，AutoDL 通常提供了便于学术访问的网络设置 `source /etc/network_turbo`，你可以先启用它，再用 `huggingface-cli` 把模型下载到数据盘目录。

注意：我们需要 **BF16** 格式的模型用于 **KT** 微调，如果你下载的是 FP8 或者其他格式，请参考 [DeepSeek-V3 转化脚本](#) 进行转换。

注意使用版本

```
pip install -U huggingface_hub==0.34.0
huggingface-cli download --resume-download Qwen/Qwen3-235B-A22B-Instruct-2507 --local-dir
/root/autodl-tmp/Qwen3-235B-A22B-Instruct-2507-BF16
```

3. LoRA 微调：用 KTransformers 先把“显存不够”问题处理掉，再用 LLaMA-Factory 写实验配方

对于KT+llamafactory来说，微调命令不需要改来改去，完整运行命令为：

```
cd LLaMA-Factory
USE_KT=1 llamafactory-cli train examples/train_lora/qwen3moe_lora_sft_kt.yaml
```

你主要需要修改 YAML 配置文件。其中 YAML 里需要明确启用 KTransformers 优化，并指定放置策略文件 `kt_optimize_rule`，具体参考 `examples/train_lora/qwen3moe_lora_sft_kt.yaml`（如下图）。

```
qwen3moe_lora_sft_kt.yaml
1  ### model
2  model_name_or_path: /mnt/data2/models/Qwen3-235B-A22B-Instruct-2507
3  trust_remote_code: true
4
5  ### method
6  stage: sft
7  do_train: true
8  finetuning_type: lora
9  lora_rank: 8
10 lora_target: all
11
12 ### dataset
13 dataset: identity, alpaca_en_demo
14 template: qwen3_nothink
15 cutoff_len: 2048
16 max_samples: 100000
17 overwrite_cache: true
18 preprocessing_num_workers: 16
19 dataloader_num_workers: 4
20
21 ### output
22 output_dir: /mnt/data/lpl/test_adapter/Kdemo_qwen
23 logging_steps: 10
24 save_steps: 200
25 plot_loss: true
26 overwrite_output_dir: true
27 save_only_model: false
28 report_to: none # choices: [none, wandb, tensorboard, swanlab, mlflow]
29
```

```

30  ### train
31  per_device_train_batch_size: 1
32  gradient_accumulation_steps: 8
33  learning_rate: 1.0e-4
34  num_train_epochs: 3
35  lr_scheduler_type: cosine
36  warmup_ratio: 0.1
37  bf16: true
38  ddp_timeout: 180000000
39  resume_from_checkpoint: null
40
41  ### ktransformers
42  use_kt: true # use KTransformers as LoRA sft backend
43  kt_optimize_rule: examples/kt_optimize_rules/Qwen3Moe-sft-amx.yaml
44  cpu_infer: 32
45  chunk_size: 8192
46
47  ### eval
48  # eval_dataset: alpaca_en_demo
49  # val_size: 0.1
50  # per_device_eval_batch_size: 1
51  # eval_strategy: steps
52  # eval_steps: 500
53

```

训练完成后，LoRA 结果会保存在 output_dir 指定目录中，一般是 safetensors 形式并附带 adapter 配置文件，后续推理直接加载这个目录即可。

4. 推理验证（可选步骤）：用 LLaMA-Factory 快速交互确认效果

当你刚微调完，稳妥的做法是先用 **LLaMA-Factory 快速交互验证**：加载基座模型 + LoRA 适配器，直接对话几轮，确认你想要的风格/能力确实出现了。这一步不追求极致吞吐，只追求确认结果可信、可复现。


```

cd LLaMA-Factory
llamafactory-cli chat examples/inference/qwen3moe_lora_sft_kt.yaml

```

如下图所示，你可以写一份推理 YAML，指定基座模型路径与 adapter 目录，并把 infer_backend 设为 ktransformers，同时确保 use_kt 与 kt_optimize_rule 和训练时保持一致，这样加载与计算路径不会错位。

```

inference >  qwen3moe_lora_sft_kt.yaml
1  model_name_or_path: Qwen/Qwen3-235B-A22B-Instruct-2507
2  adapter_name_or_path: saves/Kllama_Qwen3MoE_235bA22b
3  template: qwen3_nothink
4  infer_backend: ktransformers # choices: [huggingface, vllm, sglang, ktransformers]
5  trust_remote_code: true
6
7  use_kt: true # use KTransformers as LoRA sft backend to inference
8  kt_optimize_rule: examples/kt_optimize_rules/Qwen3Moe-sft-amx.yaml
9  cpu_infer: 32
10 chunk_size: 8192

```


你如果还想用 LLaMA-Factory 进行批量评测，也可以直接用 LLaMA-Factory 起一个 API 服务，配置文件仍然是同一份推理 YAML。这样你可以很快把模型接到自己的评测脚本里，把结果沉淀成论文实验表格或 demo 展示。

```
# YAML 文件和上面保持一致
API_PORT=8000 llamafactory-cli api examples/inference/qwen3moe_lora_sft_kt.yaml
```

5. 快速+稳定推理测评：用 SGLang 把模型变成稳定 API，并支持加载 LoRA

如果你想更快、更稳定的进行批量推理测评（比如测试大量benchmark），建议用 SGLang 起服务。KTransformers 会在 SGLang 的 server 侧接管关键计算路径，你主要需要做三件事：准备好 LoRA 适配器（做格式转换）、准备好 CPU 侧权重（做 INT8 量化）、然后用 launch_server 起服务并在参数里打开 KT 与 LoRA。

1. 你需要让 SGLang 识别 LoRA adapter，先做一次转换

```
# 你需要先转换你的checkpoint
cd sglang
python convert_lora.py <YOUR_LORA_ADAPTER_PATH>
```

2. 如果你的模型是 BF16，且你希望在 CPU 侧更省内存/更快推理，可以把 CPU 权重量化成 INT8 并指定输出目录。或者面向 Minimax M2/M2.1 以及 Kimi K2，可以使用其原精度进行推理（FP8，INT4），具体可以参考 KTransformers V0.5.0 及后续的更新（需要同步安装 V0.5.0 及以上 KTransformers）。

```
cd ktransformers/kt-kernel
python scripts/convert_cpu_weights.py \
  --input-path <PATH_TO>/Qwen3-30B-A3B-Instruct-2507 \
  --input-type bf16 \
  --output <PATH_TO>/Qwen3-30B-A3B-Instruct-2507-INT8 \
  --quant-method int8
```

3. 然后在 launch_server 时通过 `--kt-weight-path` 指向量化后的权重目录，通过 `lora-paths` 指向 LoRA Adapter。其他 SGLang 启动的 advanced 参数设置，请参考 [SGLang Tutorial](#)。

```
python -m sglang.launch_server \
  --host 0.0.0.0 \
  --port 10103 \
  --model <PATH_TO>/Qwen3-30B-A3B-Instruct-2507 \
  --mem-fraction-static 0.7 \
  --chunked-prefill-size 2048 \
  --served-model-name Qwen3-30B-A3B-Instruct-2507 \
  --tensor-parallel-size 1 \
  --kt-method AMXINT8 \
  --kt-weight-path <PATH_TO>/Qwen3-30B-A3B-Instruct-2507-INT8 \
  --kt-cpuinfer 64 \
  --kt-threadpool-count 2 \
  --kt-num-gpu-experts 1 \
  --enable-lora \
```

```
--lora-paths lora0=<YOUR_ADAPTER_PATH> \  
--max-loras-per-batch 1 \  
--lora-backend triton  
# 上面最后这四行是为了推理lora后的模型，如果只推理原模型，删除上面四行
```

运行SGLang Server成功的效果图如下：

```
--enable-lora \  
[2025-12-24 07:33:02] Memory pool end. avail mem=11.88 GB  
[2025-12-24 07:33:02] Capture cuda graph begin. This can take up to several minutes. avail mem=11.36 GB  
[2025-12-24 07:33:02] Capture cuda graph bs [1, 2, 4, 8, 12, 16, 24, 32]  
Capturing batches (bs=32 avail_mem=11.31 GB): 0% | 0/8 [00:00<?, ?it/s]  
[2025-12-24 07:33:04] Using MoE kernel config. Performance might be sub-optimal! Config file not found at /home/lpl/sclang/python/sclang/srt/layers/moe/fused_moe_triton  
/configs/triton_3_5_1/E=1,N=1536,device_name=NVIDIA_GeForce_RTX_4090.json, you can create them with https://github.com/sql-project/sclang/tree/main/benchmark/k  
triton  
[2025-12-24 07:33:04] Using MoE kernel config with down_moe=False. Performance might be sub-optimal! Config file not found at /home/lpl/sclang/python/sclang/srt/layers/moe/fuse  
d_moe_triton/configs/triton_3_5_1/E=1,N=1536,device_name=NVIDIA_GeForce_RTX_4090_down.json, you can create them with https://github.com/sql-project/sclang/tree/main/benchmark/k  
ernels/fused_moe_triton  
Capturing batches (bs=1 avail_mem=10.96 GB): 100% | 8/8 [00:10<00:00, 1.25s/it]  
[2025-12-24 07:33:13] Capture cuda graph end. Time elapsed: 10.60 s. mem usage=0.45 GB, avail mem=10.91 GB.  
[2025-12-24 07:33:13] max_total_num_tokens=80480, chunked_prefill_size=2048, max_prefill_tokens=16384, max_running_requests=2048, context_len=262144, available_gpu_mem=10.91 GB  
[2025-12-24 07:33:14] INFO: Started server process [1506325]  
[2025-12-24 07:33:14] INFO: Waiting for application startup.  
[2025-12-24 07:33:14] Using default chat sampling params from model generation config: {'repetition_penalty': 1.0, 'temperature': 0.7, 'top_k': 20, 'top_p': 0.8}  
[2025-12-24 07:33:14] Using default chat sampling params from model generation config: {'repetition_penalty': 1.0, 'temperature': 0.7, 'top_k': 20, 'top_p': 0.8}  
[2025-12-24 07:33:14] INFO: Application startup complete.  
[2025-12-24 07:33:14] INFO: Uvicorn running on http://0.0.0.0:8173 (Press CTRL+C to quit)  
[2025-12-24 07:33:15] INFO: 127.0.0.1:45376 - "GET /model_info HTTP/1.1" 200 OK  
[2025-12-24 07:33:15] Prefill batch, #new-seq: 1, #new-token: 6, #cached-token: 0, token usage: 0.00, #running-req: 0, #queue-req: 0,  
[2025-12-24 07:33:19] INFO: 127.0.0.1:45392 - "POST /generate HTTP/1.1" 200 OK  
[2025-12-24 07:33:19] The server is fired up and ready to roll!
```

4. 接下来可以通过 HTTP request 或者其他方式来调用刚才 SGLang Server 启动的服务，来进行批量推理。下面给出一个简单的示例：

```
from openai import OpenAI  
  
client = OpenAI(base_url="http://localhost:10103/v1", api_key="EMPTY") # 这里的10103需要对齐  
SGLang Server的port  
  
prompt = "使用 c++, python 和 rust, 写一个快速排序。</think>\n"  
for i in range(1):  
    resp = client.completions.create(  
        model="Qwen3-30B-A3B-Instruct-2507",  
        prompt=prompt,  
        max_tokens=256,  
    )  
    print(resp.choices[0].text)
```

Customize/Advanced KT 设置

这一节默认你已经能按前文跑通一次“LoRA 微调 → 推理验证 → SGLang 服务”。这里不教你从算法角度改训练超参（比如 lr、scheduler 等），只讲跟资源占用/工程交付强相关、且主要落在 **KTransformers (KT)** 的关键点。

暴露给用户侧的配置文件入口

1. 微调入口 (LLaMA-Factory YAML) `examples/train_lora/qwen3moe_lora_sft_kt.yaml`

对于 KT 来说，你主要关心里面的：use_kt / kt_optimize_rule / cpu_infer / chunk_size，其余的可以参考你的算法需求进行修改（比如学习率，rank 大小等）

2. 放置策略入口（由 kt_optimize_rule 指向） `examples/kt_optimize_rules/*.yaml`

这决定了“哪些算子/专家放 GPU、哪些放 CPU、用什么 CPU 后端（AMX 等）”

3. 推理交付入口 (SGLang launch_server 命令)

SGLang 相关: `--mem-fraction-static / --chunked-prefill-size / --max-running-requests`
(稳定性&显存)

KT 相关: `--kt-method / --kt-weight-path / --kt-cpuinfer / --kt-num-gpu-experts` (“显存不够”怎么转化成“CPU+策略能解决”)

4. 两个核心工具脚本 (为了适配各架构下的内容)

LoRA 适配器转换: `sglang/python convert_lora.py <YOUR_LORA_ADAPTER_PATH>`

CPU 权重量化 (INT8): `ktransformers/kt-kernel/scripts/convert_cpu_weights.py ... --quant-method int8`

降低微调显存：从 KT 微调侧入手

先改 kt_optimize_rule

kt_optimize_rule 本质是“放置策略”：把 MoE 的重计算（尤其专家相关）从 GPU 显存压力里挪走，把 GPU 留给更刚需、更难搬的部分（例如 attention 的关键路径），从而把“显存上限”变成“CPU 内存 + 放置策略”的工程问题。

rule 文件的命名规则，便于进行查找

你会看到类似：

- *-sft-*: 给微调用策略（不要拿推理策略直接来训练）
- *-amx-*: CPU 用 AMX（有 AMX 的机器优先选）
- *-multi-gpu-X*: 按 GPU 数做模型并行/切分（X=你可用的 GPU 数）

你不需要一上来读懂每个 match/replace；先用“正确的文件名”把大方向对齐，再做小调。

微调 YAML 里，KT 相关只建议动这 4 个（其余是微调 lora 算法的超参）

```
use_kt: true                # 是否启用 KT 后端
kt_optimize_rule: ...       # 放置策略，参考 https://github.com/kvcache-ai/ktransformers/blob/main/doc/en/KTransformers%20Full%20Introduction%20for%20Motivation%20and%20Practice.pdf
cpu_infer: ...              # CPU 侧并行/推理相关的开关或力度（按项目默认先跑通再改）
chunk_size: ...             # 训练/计算的分块相关（优先用于解决 OOM / 峰值显存问题）
```

降低推理显存：从 SGLang + KT 推理（服务）侧入手

SGLang 推理时的显存压力主要来自两类：

- 1) KV cache（跟并发、上下文长度强相关）
- 2) 你选择放在 GPU 的那部分算子/专家（KT 的 gpu experts 等）

可以参照下面顺序来减少显存需求。

先处理 KT 的 GPU 常驻：kt-num-gpu-experts（直接影响显存）

`--kt-num-gpu-experts`：你放在 GPU 上的专家数量（或同类含义的 GPU 侧专家驻留程度）

越大：推理可能更快，但显存更高

越小：更省显存，但更多计算会落到 CPU 侧（吞吐可能下降）

这是“用 CPU 内存/算力换 GPU 显存”的核心杠杆之一

再处理 KV cache 峰值：chunked prefill + 并发上限

A. 如果你遇到prefill 阶段 OOM（长 prompt/大 batch 更常见）

- 优先把 `--chunked-prefill-size` 调小（例如 4096 或 2048）
- 代价：长 prompt 的 prefill 会变慢一些，但更稳

B. 如果你遇到decode 阶段 OOM（并发高更常见）

- 优先降低 `--max-running-requests`
- 这本质是在限制“同时持有 KV cache 的活跃请求数”，通常是最直接的稳定性旋钮

（可选）如果你做压测/跑大 benchmark，希望更可控

- 结合 `--max-total-tokens` / `--max-prefill-tokens` 这类“总 token 上限”去兜底，避免极端请求把显存吃穿

最后处理显存静态预留：mem-fraction-static

- `--mem-fraction-static` 控制 SGLang 在 GPU 上的静态内存分配比例（常用于让缓存/调度更稳定）
- 经验上：先用中等值跑通（例如 0.7~0.9），再根据 OOM 类型微调：
如果仍然 OOM：往下调一点点试（避免一次调太猛）
如果报“not enough memory / cache 不够”之类：往上调一些

最小改动的排障/调参顺序

1. 先确认是 prefill OOM 还是 decode OOM（看日志阶段）
2. prefill OOM → 调小 `--chunked-prefill-size`（4096/2048 起步）
3. decode OOM → 降低 `--max-running-requests`（先砍一半观察）
4. 仍然显存紧 → 降低 `--kt-num-gpu-experts`（把 GPU 专家驻留降下来）
5. CPU 内存/带宽吃紧 → 用 `convert_cpu_weights.py` 做 INT8，减小 CPU 侧权重占用，通常也能让 CPU 推理更友好
6. 最后再微调 `--mem-fraction-static`，让服务在你的压测目标下更稳定

说明：这份模板的目标是“更稳、更省显存”，不是追求极致速度。你一旦跑稳了，再逐步把 `chunked-prefill-size`、`max-running-requests`、`kt-num-gpu-experts` 往上试，找到你机器最合适的区间即可。