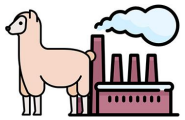
More Open Source Integration

趋境科技 APPROACHING.AI

SGL [Feature] KTransformers Integration to Support CPU/GPU Hybrid Inference for MoE Models #11425 https://github.com/sgl-project/sglang/issues/11425

Inference – Integrated into SGLang for wider model support and multi-GPU acceleration


LLaMA-Factory
Easy and Efficient LLM Fine-Tuning

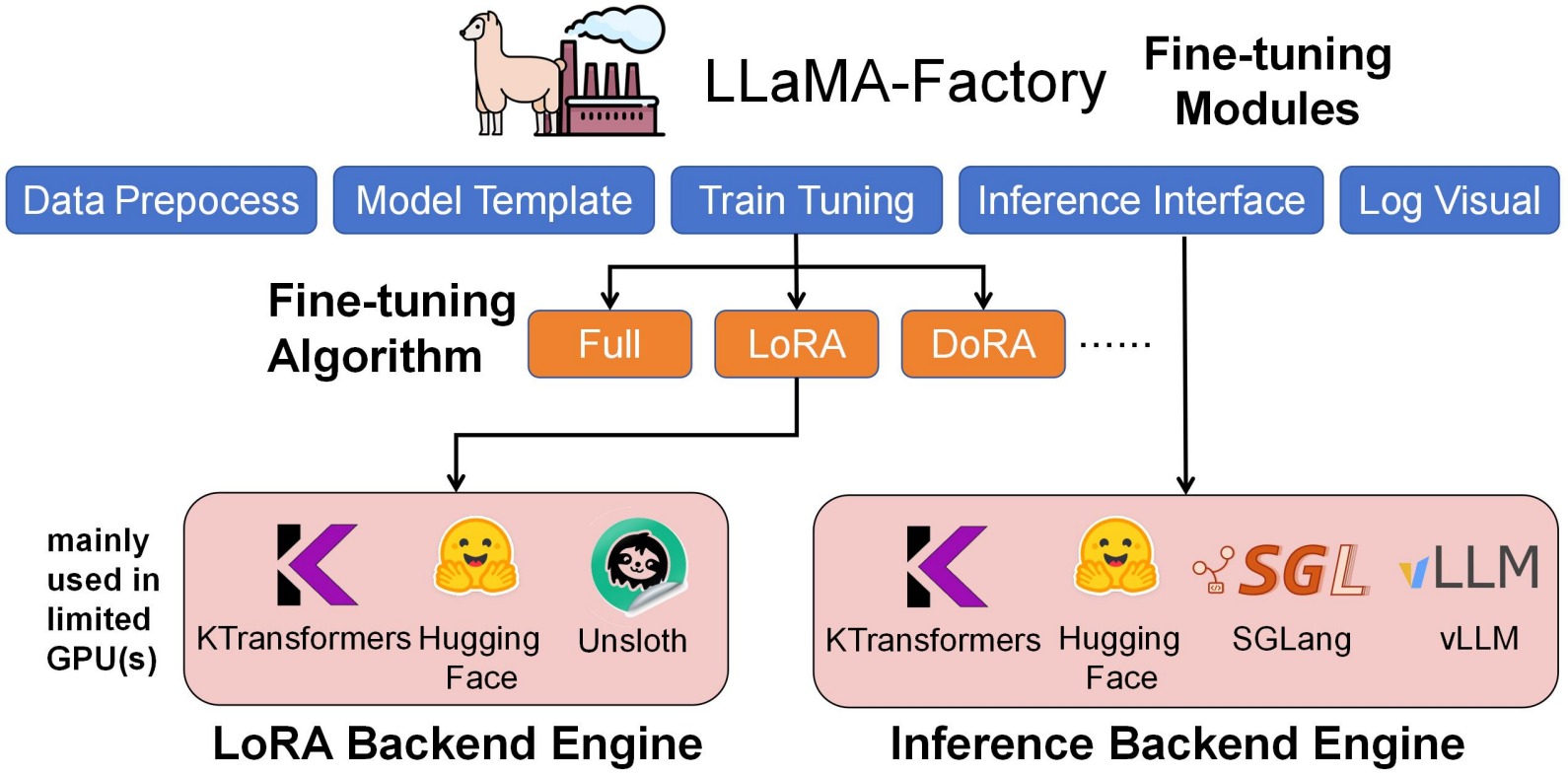
[Roadmap] Integration of KTransformers as a LoRA Fine-Tuning Backend for LLaMA-Factory #9266 https://github.com/hiyouga/LLaMA-Factory/issues/9266

FineTuning – Integrated into LLaMA-Factory for local fine-tuning

You will be able to fine-tuning 671B DeepSeek and 1TB Kimi K2 locally with consumer GPUs + server CPUs

KTransformers SFT (LoRA)

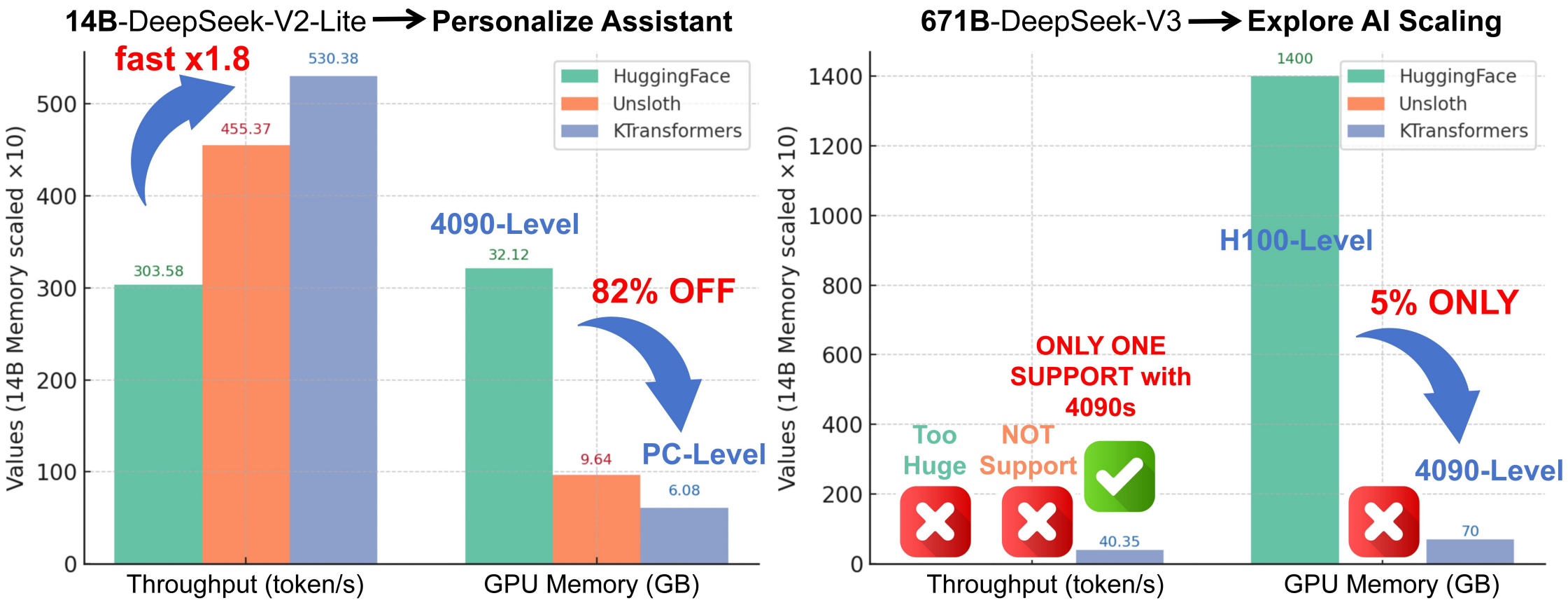
Overview Framework



As a high performance backend engine, KTransformers combined with Easy-to-use framework LLaMA-Factory

KTransformers SFT (LoRA)

Performances -- Overview



Test three backend for LoRA Fine-Tuning with LLaMA-Factory

KTransformers SFT (LoRA)

Performances -- Stylized&Benchmark

趋境科技 APPROACHING.AI

DeepSeek-V2-Lite-14B

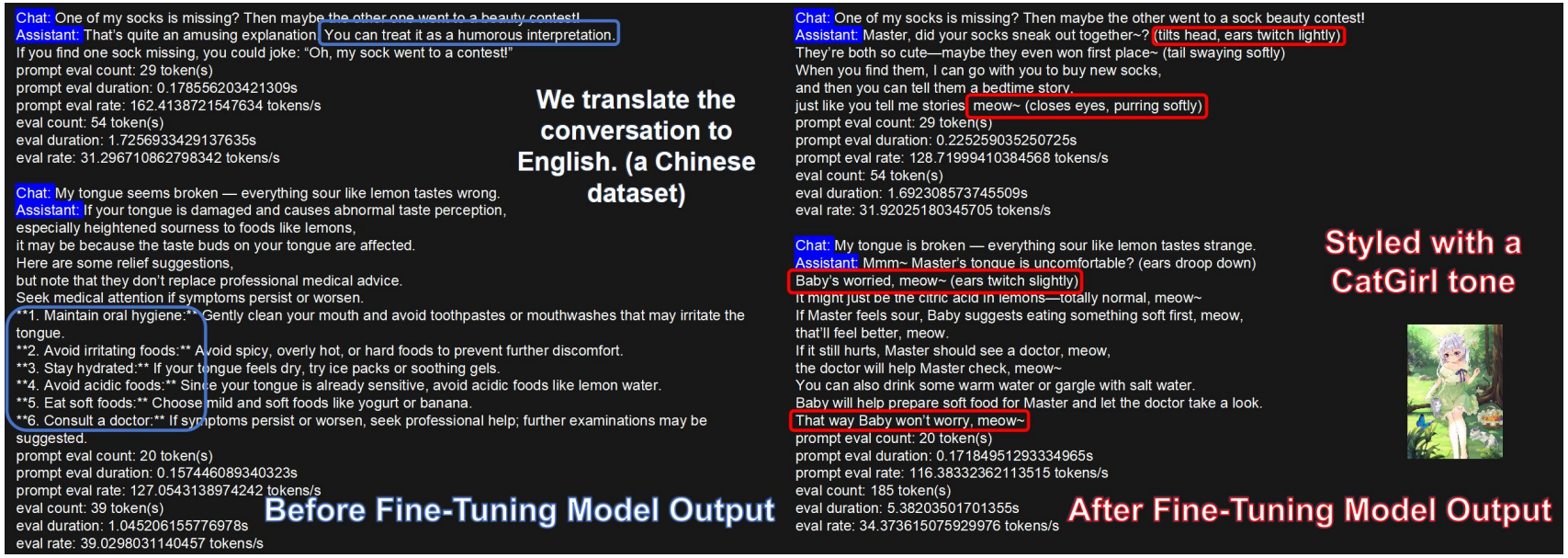
- 5GB GPU VRAM
- 30GB CPU RAM
- End-to-End 227.6 token/s

DeepSeek-V3/R1-671B

- 70GB GPU VRAM
- 1.2TB CPU RAM
- End-to-End 40.35 token/s

Kimi-K2-1T

- 81GB GPU VRAM
- 2.1TB CPU RAM
- End-to-End 36.55 token/s

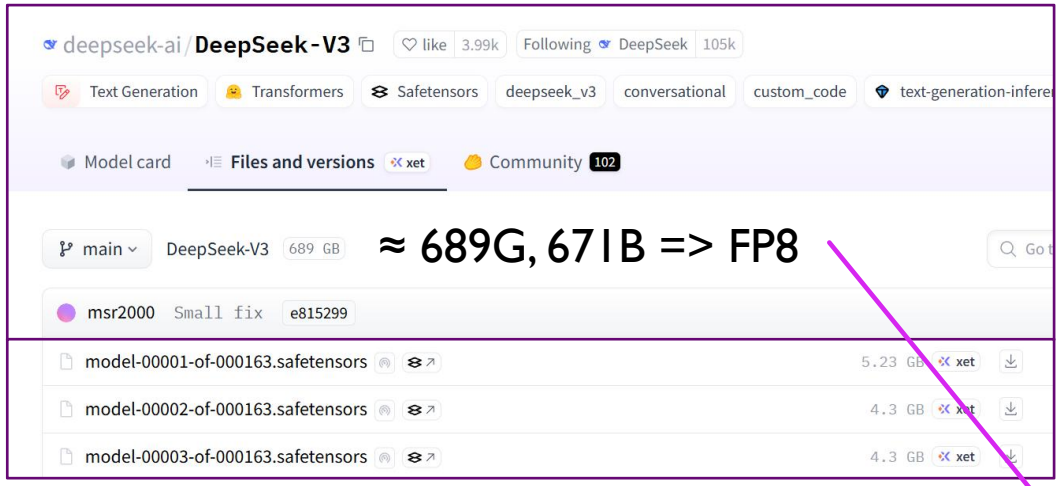


AfriMed-QA (SAQ)	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
V2-Lite (no LoRA)	13.58	11.12	9.10	7.23	22.48	7.81	11.73
KT-LoRA fine-tuned V2-Lite	**35.90**	**27.63**	**22.99**	**19.15**	**35.25**	**17.50**	**28.44**
V3 base (no LoRA)	12.75	10.27	8.05	5.99	20.33	5.65	10.11
KT-LoRA fine-tuned V3	**42.42**	**34.12**	**28.95**	**24.54**	**41.97**	**22.37**	**33.28**

Customize your KTransformers-FT

Customize model

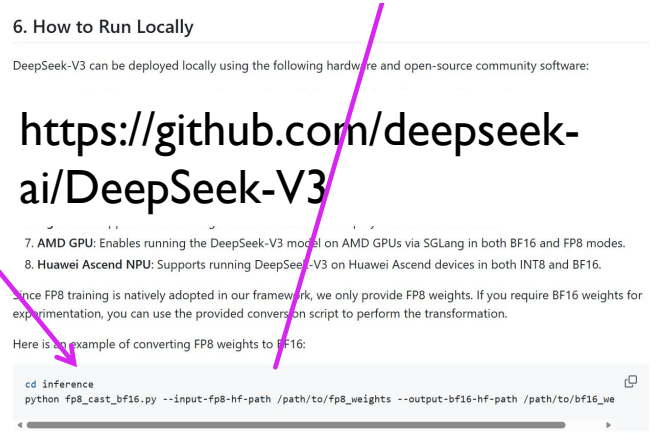
model_name_or_path: **must be BF16 model**




```
### model
model_name_or_path: opensourcerelease/DeepSeek-V3-bf16
trust_remote_code: true

### method
stage: sft
do_train: true
finetuning_type: lora
lora_rank: 8
lora_target: all
```

Put the model path after convert



Similarly,

Kimi-K2 is INT4 format,

should convert to BF16,

then fine-tuning with KT.

Customize your KTransformers-FT

Customize LoRA & Train

Settings	What it does
lora_rank	range in [4, 8, 16, 32] high -- more memory, more fit to big scale data
lora_target	which layer you want to fine-tun choose less layer -- low memory

```
### method
stage: sft
do_train: true
finetuning_type: lora
lora_rank: 8
lora_target: all
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000
resume_from_checkpoint: null
```

Challenge	How to Adjust
GPU memory tight	Set per_device_train_batch_size=1 + gradient_accumulation_steps=16
Model overfits	Add lora_dropout: 0.1 + reduce `num_train_epochs` to 2

Customize your KTransformers-FT

Customize Dataset

Step1: Construct your own data, fit with the format as follows

LLaMA-Factory / data / alpaca_en_demo.json

Code | Blame | 4997 lines (4997 loc) · 840 KB

```
[
  {
    "instruction": "Describe a process of making crepes.",
    "input": "",
    "output": "Making crepes is an easy and delicious process! Here are step-by-step instructions
  },
  {
    "instruction": "Transform the following sentence using a synonym: The car sped quickly.",
    "input": "",
    "output": "The car accelerated rapidly."
  },
```

Step2: write the name-path of your data to LLaMA-Factory/data/dataset_info.json

LLaMA-Factory / data / dataset_info.json

Code | Blame | 734 lines (734 loc) · 17 KB

```
{
  "identity": {
    "file_name": "identity.json"
  },
  "alpaca_en_demo": {
    "file_name": "alpaca_en_demo.json"
  },
  "alpaca_zh_demo": {
    "file_name": "alpaca_zh_demo.json"
  },
```

Step3:

```
### dataset
dataset: identity
template: deepseek
cutoff_len: 2048
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
dataloader_num_workers: 4

### output
output_dir: saves/Kllama_deepseekV3
logging_steps: 10
save_steps: 500
plot_loss: true
overwrite_output_dir: true
save_only_model: false
report_to: none  # choices: [none, wandb, tensorboard, swanlab, mlflow]
```

replace the default name with your data name

template: must fit the pre-trained model

Supported Models

Model	Model size	Template
Baichuan 2	7B/13B	baichuan2
BLOOM/BLOOMZ	560M/1.1B/1.7B/3B/7.1B/176B	-
ChatGLM3	6B	chatglm3
Command R	35B/104B	cohere
DeepSeek (Code/MoE)	7B/16B/67B/236B	deepseek
DeepSeek 2.5/3	236B/671B	deepseek3

cutoff_len: truncates long texts

max_samples: set 100 for debug, None for full training

Customize your KTransformers-FT

Customize KT-Optimize

What is KT Optimize Rule?

Take a example,

Regular Expression

```
- match:
    name: "^model\\.layers\\.(0|[1-9]|[12][0-9])\\.mlp\\.experts$"
  replace:
    class: ktransformers.operators.experts.KTransformersExperts     # cust
    kwargs:
      prefill_device: "cuda:0"
      prefill_op: "KExpertsTorch"
      generate_device: "cpu"
      generate_op: "KSFTExpertsCPU"
      out_device: "cuda:0"
      backend: "AMXInt8" # or "AMXBF16" or "llamafile" (default)
  recursive: False # don't recursively inject submodules of this module
```

Expert-Parallel Operator with SFT

Different layer place on different cuda device

```
- match:
    name: "^model\\.layers\\.([3456][0-9])\\.mlp\\.experts$"
    out_device: "cuda:1"
```

KTransformers offers **high-performance operators,** which **replace** the original model operators **following our optimization rules**.

```
### ktransformers
use_kt: true # use KTransformers as LoRA sft backend
kt_optimize_rule: examples/kt_optimize_rules/DeepSeek-V3-Chat-sft-amx-multi-gpu.yaml
cpu_infer: 32
chunk_size: 8192
```

KT Support Operators (Partly)

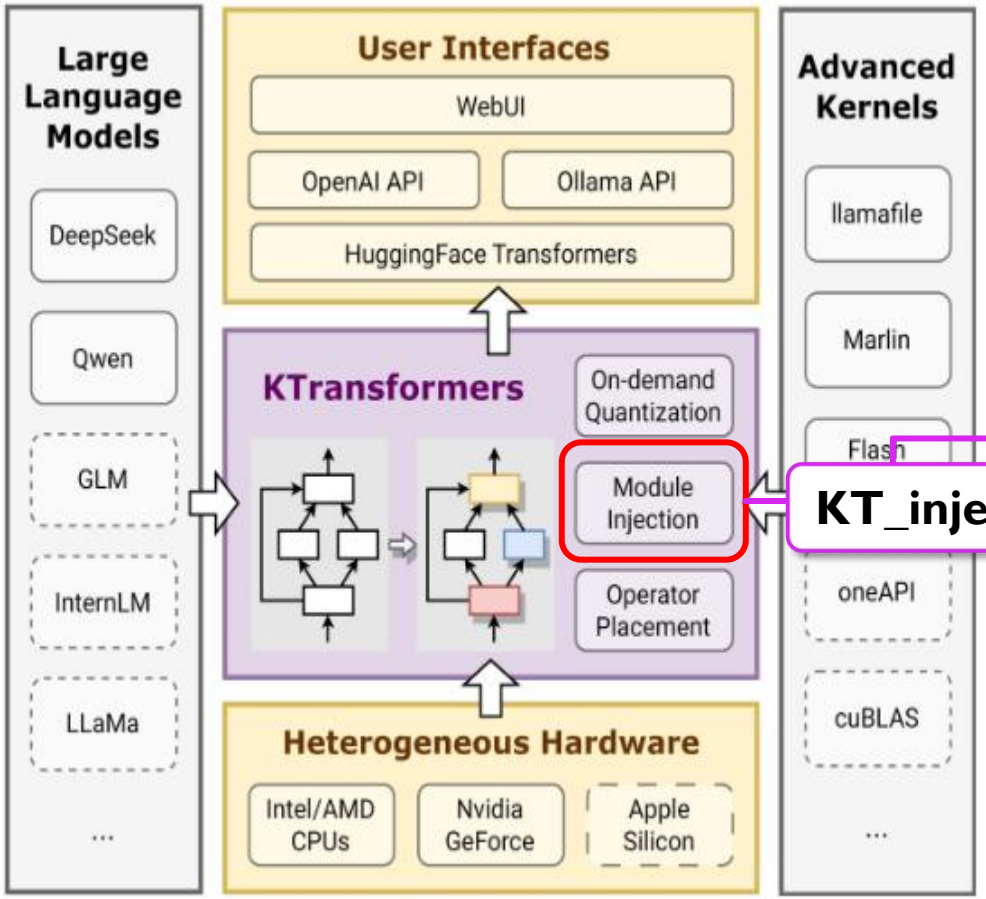
match	replace	backends	descriptions
Linear	KTransformersLinear	KLinearMarlin	Marlin as backend
		KLinearTorch	pytorch as backend
		KLinearCPUInfer	llamafile as backend
		KLinearFP8	Triton fp8_gemm kernel. Requires GPU be able to caluculate fp8 data
experts	KTransformersExperts	KExpertsTorch	pytorch as backend
		KExpertsMarlin	Marlin as backend
		KExpertsCPU	llamafile as backend
Attention	KDeepseekV2Attention	KDeepseekV2Attention	MLA implementation
MoE	KMistralSparseMoEBlock	KQwen2MoeSparseMoeBlock	MoE for Qwen2
	KDeepseekV2MoE	KDeepseekV2MoE	MoE for DeepseekV2
Model	KQwen2MoeModel	KQwen2MoeModel	Model for Qwen2
	KDeepseekV2Model	KDeepseekV2Model	Model for DeepseekV2
RoPE	RotaryEmbedding	RotaryEmbedding	RoPE module
	YarnRotaryEmbedding	YarnRotaryEmbedding	RoPE module

KTransformers SFT (LoRA)

Tech Part I : KT-Attention (KTLinearLora)

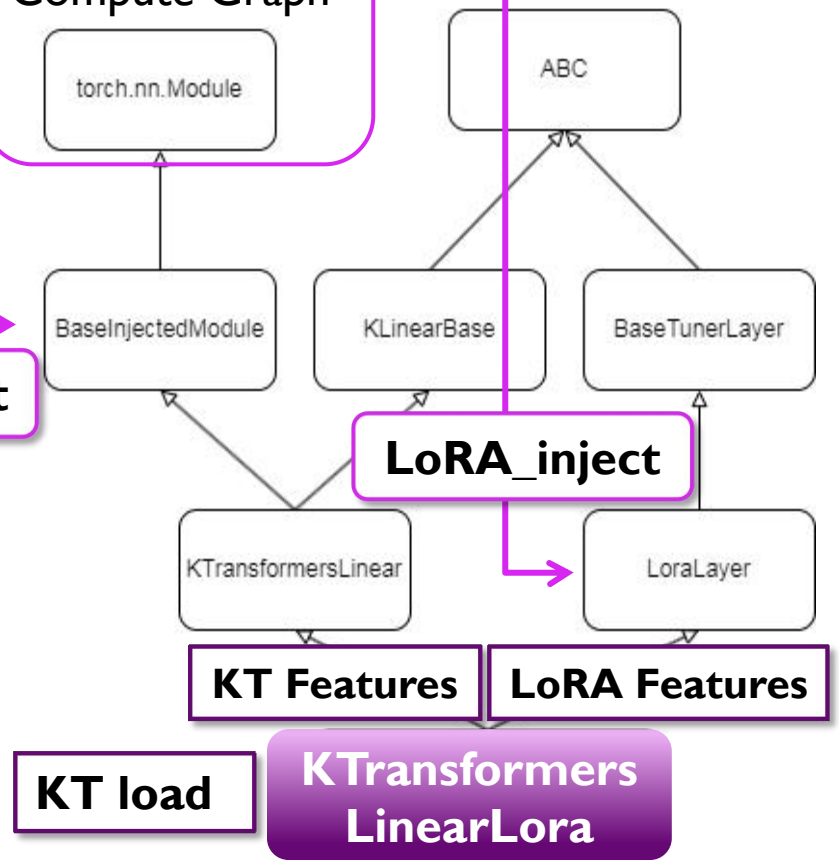
趋境科技 APPROACHING.AI

Retaining the inserted features of both KT and LoRA



KT_inject

Capture by Torch Compute Graph



Linear LoRA target_modules

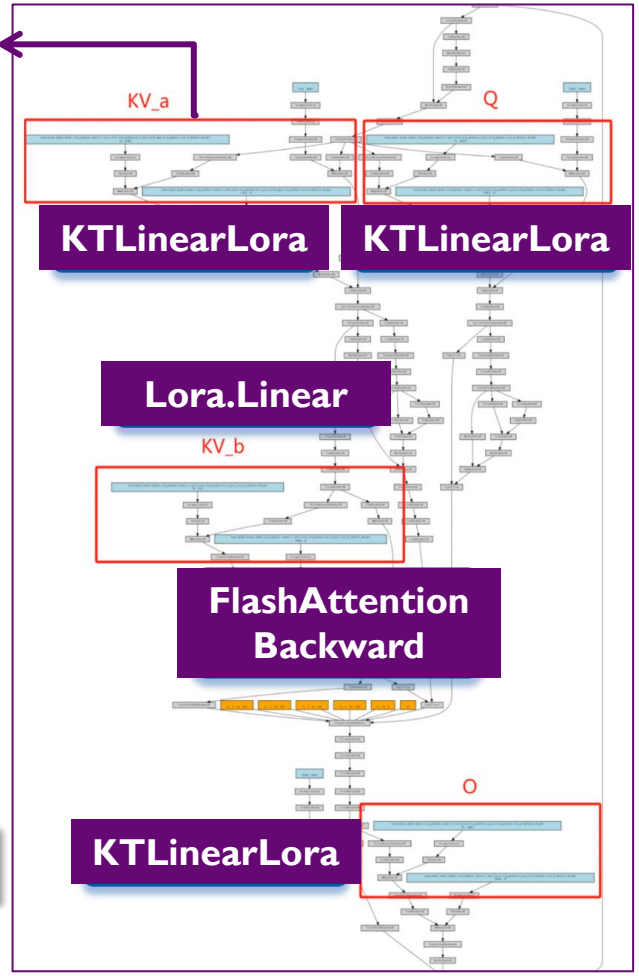
LoRA_inject

KT Features

LoRA Features

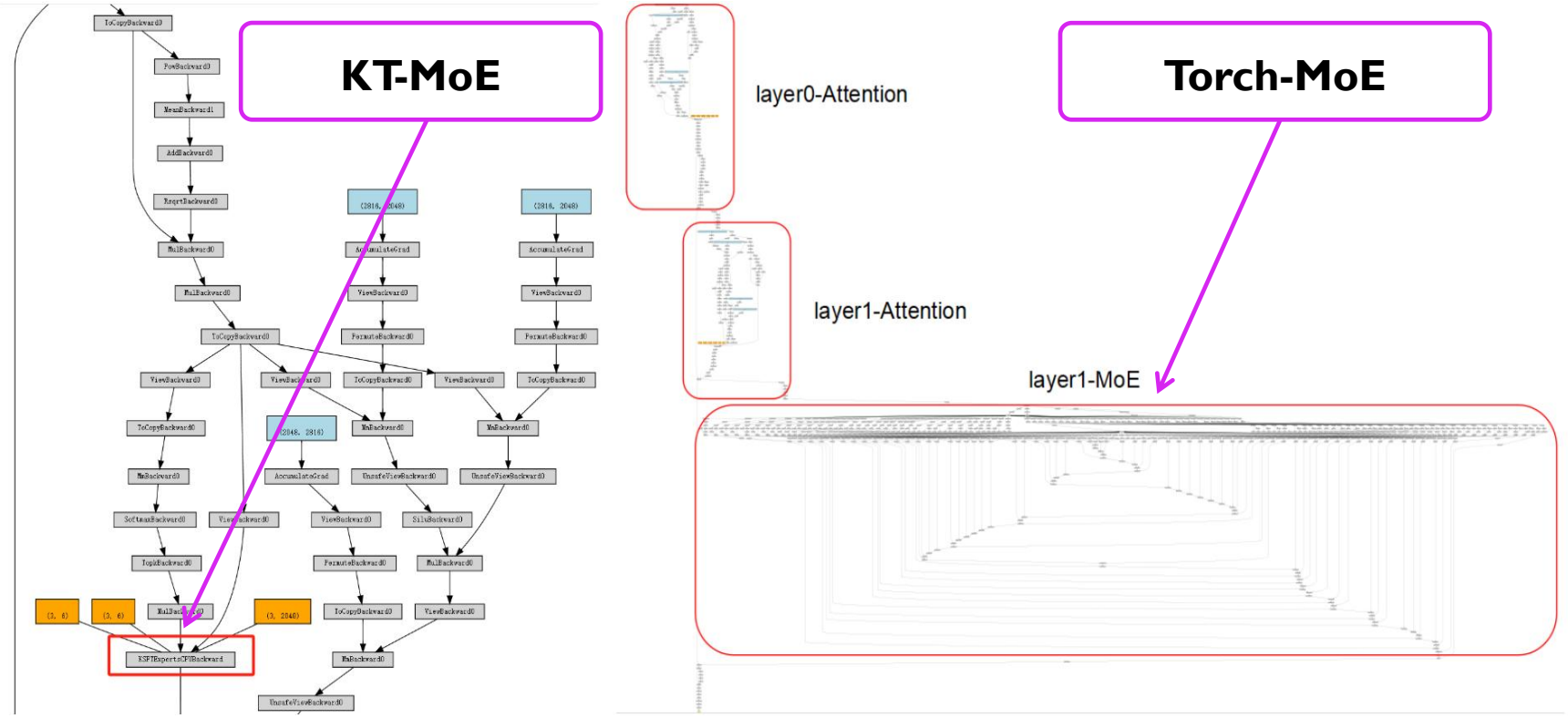
KT load

KTransformers LinearLora



In torch compute graph:

Compute the backward of MoE in CPU, not seen in torch compute graph.



- Support AMX/llamafile
- Support NUMA
- Support forward cache

KTransformers SFT (LoRA)

Tech Part III: KT-multi-gpu LoRA

DeepSeek-V3-671B requires 70G VRAM in KT， needs to place on 2 or more RTX 4090

Construct Class KTrainer & KAccelerate

- Avoid transformers carry model to single-gpu, keep KT placement
- **KTrainer:** Use ModelParallel, forbid DataParallel for multi-gpu
- **KAccelerate:** loss move to cuda:0, other tensor remain in multi-gpu

Test Result

MoE： AMX+Intel(R) Xeon(R) Platinum 8488C
+2 RTX4090 (48G VRAM)

	TFLOPS	计算时间/层
Forward	9.53	50.6ms
Backward	11.09	67.4ms