







赞同 51

分享

揭秘NVIDIA大模型推理框架: TensorRT-LLM



关注

51 人赞同了该文章

导读 大家好,我是来自 NVIDIA 的周国峰⁺,今天给大家带来的是关于TensorRT-LLM 推理框架方案的介绍。

介绍的主要内容分为如下几部分:

- 1. TensorRT-LLM+ 的产品定位
- 2. TensorRT-LLM 的重要特性
- 3. TensorRT-LLM 的使用流程
- 4. TensorRT-LLM 的推理性能
- 5. TensorRT-LLM 的未来展望
- 6. 问答环节

分享嘉宾 | 周国峰 NVIDIA DevTech 研发经理

编辑整理 | 周思源*

内容校对 | 李瑶

出品社区 | DataFun

01TensorRT-LLM 的产品定位

TensorRT-LLM 是 NVIDIA 用于做 LLM(Large Language Model)的可扩展推理方案。该方案 是基于 TensorRT 深度学习编译框架来构建、编译并执行计算图,并借鉴了许多 FastTransformer 中高效的 Kernels 实现,然后利用 NCCL 完成设备之间的通讯。考虑到技术的发展和需求的差异,开发者还可以定制算子来满足定制需求,比如基于 cutlass⁺ 开发定制 GEMM。TensorRT-LLM 是一款致力于提供高性能并不断完善其实用性的 NVIDIA 官方推理方案⁺。

WHAT IS TENSORRT-LLM

LLM Inference

NVIDIA solution for scalable LLM Inference. It is built on

- TensorRT to leverage its deep learning compiler for building and executing LLM models/graphs
- · FasterTransformer to leverage its optimized kernels for performance
- NCCL for scalable inference, such as TP AllReduce, PP Send&Recv
- · Other components for the customizations of LLM inference, such as cutlass

NVIDIA TensorRT-LLM aims to provide the best LLM inference performance on NVIDIA GPU with great flexibility ${\bf \hat{x}}$ usability

TensorRT-LLM 已经在 GitHub 上开源,主要分为两个分支,即 Release branch 和 Dev branch。其中 Release branch 每个月更新一次,而在 Dev branch 中则会较为频繁地更新来自官方或社区中的功能,方便开发者体验、评估最新功能。下图展示了 TensorRT-LLM 的框架结构,其中除了绿色 TensorRT 编译部分和一些涉及硬件信息的 kernels 外,其他部分都是开源的。



WHAT IS TENSORRT-LLM

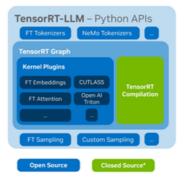
OSS

Open source software (OSS) https://github.com/NVIDIA/TensorRT-LLM

- · Stable branch for production
- Dev branch for quick exploring new features in TensorRT-LLM

Open Source

- · All Python APIs for model definition and weight loading
- All plugins/kernels except those with a risk of exposing HW information
 - FMHA, Batch Manager
- · Pre/post processing, such as tokenizers and samplers



TensorRT-LLM 还提供了类似于 Pytorch 的 API 来降低开发者的学习成本,并提供了许多预定义好的模型供用户使用。



WHAT IS TENSORRT-LLM

Python API

Pytorch-like Python API

- Great usability
 - No need to bother with memory manager, CUDA calls, etc.
 - C++ runtime for performance
- Great modularization for easier extension
 - A lot of predefined models in TensorRT-LLM ready to be deployed for production
 A lot of predefined models in TensorRT-LLM ready to be deployed hidden_states = self.embedding(_)
 - · Users can build customized models easily
 - · For performance, may need to write kernels in CUDA
 - NVIDIA can provide supports





WHAT IS TENSORRT-LLM

MGMN

Multi-GPU, Multi-Node (MGMN) is needed in today's LLM Inference

- 5 A800s with 80GB HBM are needed to fit GPT-3 175B under FP16
- · For a better perf/\$, more GPUs, even multi-node are needed

MGMN in TensorRT-LLM

- · Leverage NCCL plugins for multi-device communication
 - Long term will be OOTB in TRT
- Support TP, PP already

02TensorRT-LLM 的重要特性

TensorRT-LLM 的重要特性之一就是丰富的模型支持。TensorRT-LLM 对主流大语言模型都提供了支持,比如 Qwen(千问)就是由开发者完成的模型适配,并已经纳入官方支持。用户可以很容易地基于这些预定义的模型做扩展或定制。其二就是低精度推理⁺,TensorRT-LLM 默认采用FP16/BF16 的精度推理,并且可以利用业界的量化方法,使用硬件吞吐更高的低精度推理进一步推升推理性能。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

Model coverage

TensorRT-LLM provides users with abundant predefined models

- Baichuan, Bert, Blip2, Bloom, ChatGLM-6B, ChatGLM2-6B, Falcon, GPT, GPT-J, GPT-NeoX, LLaMA, OPT, SantaCoder, StarCoder, Qwen, etc.
 - · Can be modified and extended for customization easily
 - · TensorRT-LLM is capable of supporting every single LLM model

TensorRT-LLM supports low precision for SOTA LLM inference

- FP16/BF16 by default
- INT8/FP8/INT4 with quantization. Even lower precision in future

另外一个特性就是 FMHA(fused multi-head attention) kernel 的实现。由于 Transformer 中最为耗时的部分是 self-attention * 的计算,因此官方设计了 FMHA 来优化 self-attention 的计算,并提供了累加器分别为 fp16 和 fp32 不同的版本。另外,除了速度上的提升外,对内存的占用也大大降低。我们还提供了基于 flash attention * 的实现,可以将 sequence-length 扩展到任意长度。

知乎

首发于

人工智能算法实践

HIGHLIGHTED FEATURES IN TENSORRT-LLM

Flash Attention based FMHA for the Context Phase

FMHA (fused multi-head attention) for the context phase

- Note it is optional as flash attention could potentially affect the accuracy of some models.
- Flags (--enable-context-fmha/--enable-context-fmha-fp32-acc) when building engines. Also, it will fall back to unfused MHA when the case is not supported.
- Reduce memory consumption significantly
- No sequence-length limitation with flash attention.

如下为 FMHA 的详细信息,其中 MQA 为 Multi Query Attention,GQA 为 Group Query Attention。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

Flash Attention based FMHA for the Context Phase

FMHA for the context phase

Arch	Flash Attention	Mask Type	Data Type	Head Size	Supported Features
Ampere/Ada (SM 80/86/89)	Yes	Padding Causal SlidingWindowCausal	FP16(FP16 acc) FP16(FP32 acc) BF16 (FP32 acc only)	32/40/64/80/ 128/160/256	MQA/GQA/AliBi
Hopper (SM90)	Yes	Padding Causal SlidingWindowCausal	FP16(FP16 acc) FP16(FP32 acc) BF16 (FP32 acc only)	32/64/128/25 6	MQA/GQA/AliBi

另外一个 Kernel 是 MMHA(Masked Multi-Head Attention)。FMHA 主要用在 context phase 阶段的计算,而 MMHA 主要提供 generation phase 阶段 attention 的加速,并提供了 Volta 和 之后架构的支持。相比 FastTransformer 的实现,TensorRT-LLM 有进一步优化,性能提升高达 2x。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

MMHA for the Generation Phase

MMHA (Masked Multi-Head Attention) for the generation phase

- Compute with CUDA cores
- Volta and afterwards archs support
- FP16/BF16/FP32 are supported
- Continue to optimize its performance, up to 2x speedup compared to the one in FasterTransformer

而言,这两种量化方式的推理逻辑是相同的。对于 LLM 量化技术,一个重要的特点是算法设计和工程实现的 co-design,即对应量化方法设计之初,就要考虑硬件的特性。否则,有可能达不到预期的推理速度提升。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

Quantization

Quantization is necessary for LLM deployment to reduce the cost

- Reduce latency with low precision TensorCore
- · Improve throughput by increasing batch size

A lot of quantization methods have been developed

- QAT. Expensive
- · PTQ. More feasible than QAT

Quantization needs to be co-designed for efficient deployment on GPU

TensorRT 中 PTQ 量化步骤一般分为如下几步,首先对模型做量化,然后对权重和模型转化成TensorRT-LLM 的表示。对于一些定制化的操作,还需要用户自己编写 kernels。常用的 PTQ 量化方法包括 INT8 weight-only、SmoothQuant、GPTQ 和 AWQ,这些方法都是典型的 codesign 的方法。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

PTQ in TensorRT

Procedures for TensorRT-LLM support LLM quantization algorithms in general

- Model quantization (AMMO)
- Model/Weight convert
- Kernel development and TensorRT plugin
 - Cutlass https://github.com/NVIDIA/cutlass to develop customized GEMM/GEMV kernels

PTQ methods

· INT8 weight-only, SmoothQuant, GPTQ, and AWQ, etc.

Characterize with the co-design between algorithm and engineering

INT8 weight-only 直接把权重量化到 INT8,但是激活值还是保持为 FP16。该方法的好处就是模型存储2x减小,加载 weights 的存储带宽减半,达到了提升推理性能的目的。这种方式业界称作W8A16,即权重为 INT8,激活值为 FP16/BF16——以 INT8 精度存储,以 FP16/BF16 格式计算。该方法直观,不改变 weights,容易实现,具有较好的泛化性能。

首发于

人工智能算法实践

HIGHLIGHTED FEATURES IN TENSORRT-LLM

INT8 weight-only

INT8 weight-only

- Quantize weights only to INT8, aka W8A16
- Storage in INT8, compute in fp16/bf16
- · 2x reduction in weight storage
 - Few memory traffics
 - · High throughput with large batch sizes
- · Straightforward and easy to be implemented
- Good generalization ability

第二个量化方法是 SmoothQuant,该方法是 NVIDIA 和社区联合设计的。它观察到权重通常服从高斯分布⁺,容易量化,但是激活值存在离群点,量化比特位利用不高。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

SmoothQuant

SmoothQuant (Reference)

- Observation: easy to quantize weights, but hard to quantize activations with outliers
- SmoothQuant offline migrates the quantization difficulty from activations to weights

$$Y = (X \operatorname{diag}(s)^{-1}) \cdot (\operatorname{diag}(s)W) = \hat{X}\hat{W}$$

- Step 1. smooth activations X by dividing s>1 and anti-smooth weights by multiplying s
 to make the formula mathematically equal to XW
- · Step 2. quantize both smoothed activations and weights. W8A8

SmoothQuant 通过先对激活值做平滑操作即除以一个scale将对应分布进行压缩,同时为了保证等价性,需要对权重乘以相同的 scale。之后,权重和激活都可以量化。对应的存储和计算精度都可以是 INT8 或者 FP8,可以利用 INT8 或者 FP8 的 TensorCore 进行计算。在实现细节上,权重支持 Per-tensor 和 Per-token⁺ 的量化。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

SmoothQuant

Smooth

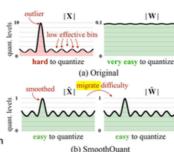
- S is a per-channel smoothing factor (vector of size C_input)
- · Smooth is performed offline

Ouantization

- · Per-tensor or per-channel for weights
- Per-tensor or per-token (also outer dimension of GEMM) for activation

SmoothQuant is a typical algorithm of HW&SW co-design

· Efficient to be implemented on GPU



就比较大,所以作者设计了一些 trick 来降低量化本身的开销,比如 Lazy batch-updates 和以相同顺序量化所有行的权重。GPTQ 还可以与其他方法结合使用如 grouping 策略。并且,针对不同的情况,TensorRT-LLM 提供了不同的实现优化性能。具体地,对 batch size 较小的情况,用 cuda core 实现;相对地,batch size * 较大时,采用 tensor core 实现。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

GPTQ

GPTQ Overview (Reference)

- · A layer-wise quantization algorithm minimizing a layer-wise reconstruction loss
 - Loss function | |Wx WqX| |
 - Weight-only quantization. Weight storage in low precision, compute in fp16, W4A16
- · Co-design for efficient deployment with GPU
 - · Lazy batch-updates to improve the compute intensity
 - · Quantize the weights of all rows in the same order

第四种量化方式是 AWQ。该方法认为不是所有权重都是同等重要的,其中只有 0.1%-1% 的权重 (salient weights) 对模型精度贡献更大,并且这些权重取决于激活值分布而不是权重分布。该方法的量化过程类似于 SmoothQuant,差异主要在于 scale 是基于激活值分布计算得到的。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

AWQ

AWQ (Activation-aware Weight Quantization, Reference)

- Weights are not equally important for LLM's performance. 0.1%-1% salient weights
- Salient weights are determined by activation distribution instead of weight distribution. (Activation-aware)
- Keeping Salient weights in fp16 can significantly improve the quantized performance, which is not hardware-efficient.
- Neither backpropagation nor (weight) reconstruction, preserves LLMs' generalization ability well.

知乎

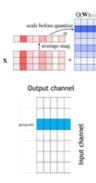
首发于 **人工智能算法实践**

HIGHLIGHTED FEATURES IN TENSORRT-LLM

AWO

AWQ (Activation-aware Weight Quantization)

- · Scale up the salient weight before quantization
 - Scalers are determined solely by activation. Similar as the smooth process in SQ, but determine scalers in another way
 - · Per-(output) channel scaling
- · Grouped quantization along the input channel direction
 - · Similar as the GPTQ but with no weight reconstruction
 - · Weight-only grouped quantization



除了量化方式之外,TensorRT-LLM 另外一个提升性能的方式是利用多机多卡推理。在一些场景中,大模型过大无法放在单个 GPU 上推理,或者可以放下但是影响了计算效率,都需要多卡或者多机进行推理。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

MGMN

Cases where MGMN is necessary or beneficial

- The model is too large to fit in a single GPU (e.g., GPT-175B)
 - · MGMN reduces the GPU memory footprint because each GPU only stores a part of model weights
- · The inference must be performed under latency constraints (e.g., online services)
 - MGMN reduces latencies by splitting the workload to multiple GPUs that are run in parallel
- The goal is to maximize throughput per GPU with no latency constraint
 - The generation phase of the inference is usually memory-bound, which means that the latency grows sub-linearly as the batch size increases
 - · Therefore, larger batch size leads to higher throughput
 - MGMN allows using larger batch size per GPU because of the reduced GPU memory footprint

TensorRT-LLM 目前提供了两种并行策略,Tensor Parallelism 和 Pipeline Parallelism。TP 是垂直地分割模型然后将各个部分置于不同的设备上,这样会引入设备之间频繁的数据通讯,一般用于设备之间有高度互联的场景,如 NVLINK。另一种分割方式是横向切分,此时只有一个横前面,对应通信方式是点对点的通信,适合于设备通信带宽较弱的场景。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

TP & PP

TensorRT-LLM implements two types of model parallelism to support MGMN

- Tensor Parallelism (TP)
 - How? Split the model vertically
 - · When? Usually, the best choice when the inter-GPU bandwidth is high (e.g., nvlink connection)
- · Pipeline Parallelism (PP)
 - · How? Split the model horizontally
 - · When? Cases when the inter-GPU bandwidth is limited (e.g., no nvlink connection)

静态batching的方法,一个batch的时延取决于 sample/request 中输出最长的那个。因此,虽然输出较短的 sample/request 已经结束,但是并未释放计算资源,其时延与输出最长的那个 sample/request 时延相同。In-flight batching 的做法是在已经结束的 sample/request 处插入新的 sample/request。这样,不但减少了单个 sample/request 的延时,避免了资源浪费问题,同时也提升了整个系统的吞吐。



HIGHLIGHTED FEATURES IN TENSORRT-LLM

In-flight batching

Maximizing GPU Utilization during LLM serving

- Replaces completed requests in the batch
 - · Evict requests after EoS & inserts a new request
- · Improves throughput, time to first token, & GPU utilization
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible through the TensorRT-LLM Batch Manger



03TensorRT-LLM 的使用流程

TensorRT-LLM 与 TensorRT的 使用方法类似,首先需要获得一个预训练好的模型,然后利用 TensorRT-LLM 提供的 API 对模型计算图进行改写和重建,接着用 TensorRT 进行编译优化,然后保存为序列化的 engine 进行推理部署。



HOW TO USE TENSORRT-LLM

Create, Build, Execute

Instantiate model and load the weights

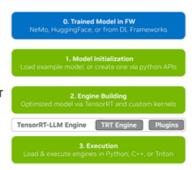
 Load pre-built models or define via TensorRT-LLM Python APIs

Build & serialize the engines

- · Compile to optimized implementations via TensorRT
- Saved as a serialized engine

Load the engines and run optimized inference!

· Execute in Python, C++, or Triton



以 Llama 为例,首先安装 TensorRT-LLM,然后下载预训练模型,接着利用 TensorRT-LLM 对模型进行编译,最后进行推理。

Run the Model

知乎 人工智能算法实践

HOW TO USE TENSORRT-LLM

Run Llama in TensorRT-LLM

Install and Build TensorRT-LLM from Github

| Sections of the content of the cont

对于模型推理[†]的调试,TensorRT-LLM 的调试方式与 TensorRT 一致。由于深度学习编译器,即TensorRT,提供的优化之一是 layer 融合。因此,如果要输出某层的结果,就需要将对应层标记为输出层,以防止被编译器优化掉,然后与 baseline 进行对比分析。同时,每标记一个新的输出层,都要重新编译 TensorRT 的 engine。



HOW TO USE TENSORRT-LLM

How to debug

Document is here

- Register the intermediate tensors as the network outputs with register_network_output API
- Use net._mark_output to set tensors as output of TRT engine, then we can print intermediate results/tensors for debugging
- Set "debug_mode=True" in GenerationSession of runtime
- Now, we can get this output afterwards, like logits here

Everytime you add a new output, please remember to rebuild the engine

对于自定义的层,TensorRT-LLM 提供了许多 Pytorch-like 算子帮助用户实现功能而不必自己编写 kernel⁺。如样例所示,利用 TensorRT-LLM 提供的 API 实现了 rms norm 的逻辑,TensorRT会自动生成 GPU 上对应的执行代码。



HOW TO USE TENSORRT-LLM

How to add a custom layer

TRT-LLM provides many fundamental pytorch-like ops

Users can use them to implement new kernels

Use rms norm as an example

*Codes for demo, full codes are here https://github.com/NVIDIA/TensorRT-LLM/blob/release/0.5.0/tensorrt_llm/functional.py#L3486-L3571

SmoothQuant 定制 GEMM 实现并封装成 plugin+后,供 TensorRT-LLM 调用的示例代码。



HOW TO USE TENSORRT-LLM

How to add a custom layer

If some special kernel cannot be supported by TRT-LLM basic ops, users can use self-defined plugins. Same as TensorRT

Use smooth-quant GEMM as an example

layer = default_trtnet().add_plugin_v2(plug_inputs, gemm_plug) layer.get_input(0).set_dynamic_range(-127, 127) return _create_tensor(layer.get_output(0), layer)

"Codes for demo, full codes are here https://github.com/NVIDIA/TensorR' LLM/blob/release/0.5.0/tensorrt_llm/quantization/functional.py#L26-L62

04TensorRT-LLM 的推理性能

关于性能、配置等细节都可以在官网看到,在此不做详细介绍。该产品从立项开始一直与国内很多大厂都有合作。通过反馈,一般情况下,TensorRT-LLM 从性能角度来说是当前最好的方案。由于技术迭代、优化手段、系统优化等众多因素会影响性能,并且变化非常快,这里就不详细展开介绍TensorRT-LLM 的性能数据。大家如果有兴趣,可以去官方了解细节,这些性能都是可复现的。

LLM PERFORMANCE

Subject to change as TensorRT LLM iterates

8x faster performance

 Custom MHAs, Inflight-batching, paged attention, quantized KV cache, & more drive inference performance

5x reduction in TCO

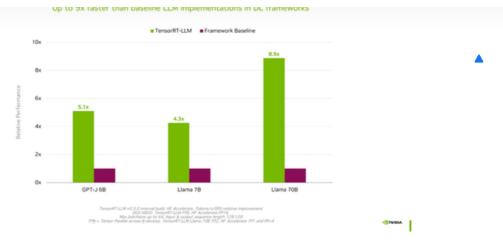
 FP8 & Inflight-batching performance allow for H100 to improve TCO significantly

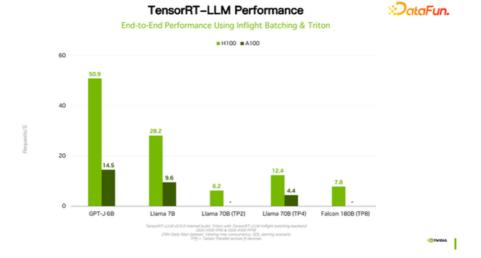
5x reduction in Energy

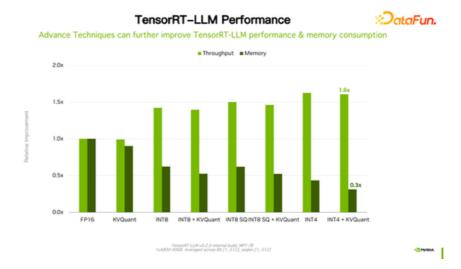
 Performance allows for reduce energy / inference allowing for more efficient use of datacenters



H800 FP8 w/ IFB in TensorRT-LLM vs A800 FP16 PyTorch







值得一提的是,TensorRT-LLM 跟自己之前的版本比,性能有持续地提升。如上图所示,在 FP16 基础上,采用了 KVQuant 后,速度一致的情况下降低了显存的使用量。使用 INT8,可以看到明显的吞吐的提升,同时显存用量进一步降低。可见,随着 TensorRT-LLM 优化技术的持续演进,性能会有持续地提升。这个趋势会持续保持。

LLM 是一个推理成本很高、成本敏感的场景。我们认为,为了实现下一个百倍的加速效果,需要算法和硬件的共同迭代,通过软硬件之间 co-design 来达到这个目标。硬件提供更低精度的量化,而软件角度则利用优化量化、网络剪枝等算法,来进一步提升性能。



THE FUTURE OF TENSORRT-LLM

Next 100x Speedup

LLM inference is extremely cost sensitive. Need another 100x speedup/cost reduction for deployment at scale

- Co-design between HW & SW is the key to achieve the target
- HW
 - · More flexible sparsity and acceleration
 - · Even low precision with higher throughput
- SW
 - Network pruning and compression
 - · Quantization/Decoding strategies

TensorRT-LLM,将来 NVIDIA 会持续致力于提升 TensorRT-LLM 的性能。同时通过开源,收集 反馈和意见,提高它的易用性。另外,围绕易用性,会开发、开源更多应用工具,如 Model zone 或者量化工具等,完善与主流框架的兼容性,提供从训练到推理和部署端到端的解决方案。



THE FUTURE OF TENSORRT-LLM

Next 100x Speedup

TensorRT-LLM

- Performance
 - Work with the community to support every single important optimization method
 - Highly-optimized kernels/Quantization/Sparsity/long-context support and optimization
- Usability
 - · Continue to improve the usability with the community. Feedback/issues are welcome
 - · Model zone/Quantization tools (AMMO)/Compatibility with DL frameworks (nemo/HF)

06问答环节

Q1: 是否每一次计算输出都要反量化? 做量化出现精度溢出怎么办?

A1:目前 TensorRT-LLM 提供了两类方法,即 FP8 和刚才提到的 INT4/INT8 量化方法。低精度如果 INT8 做 GEMM 时,累加器会采用高精度数据类型,如 fp16,甚至 fp32 以防止 overflow。关于反量化,以 fp8 量化为例,TensorRT-LLM 优化计算图时,可能动自动移动反量化结点,合并到其它的操作中达到优化目的。但对于前面介绍的 GPTQ 和 QAT,目前是通过硬编码写在 kernel中,没有统一量化或反量化节点的处理。

Q2: 目前是针对具体模型专门做反量化吗?

A2:目前的量化的确是这样,针对不同的模型做支持。我们有计划做一个更干净的api或者通过配置项的方式来统一支持模型的量化。

A3: 因为一些功能未开源,如果是自己的 serving 需要做适配工作,如果是 triton 则是一套完整的方案。

Q4:对于量化校准有几种量化方法,加速比如何?这几种量化方案效果损失有几个点?In-flight branching 中每个 example 的输出长度是不知道的,如何做动态的 batching?

A4:关于量化性能可以私下聊,关于效果,我们只做了基本的验证,确保实现的 kernel 没问题,并不能保证所有量化算法在实际业务中的结果,因为还有些无法控制的因素,比如量化用到的数据集及影响。关于 in-flight batching,是指在 runtime 的时候去检测、判断某个 sample/request 的输出是否结束。如果是,再将其它到达的 requests 插进来,TensorRT-LLM 不会也不能预告预测输出的长度。

Q5: In-flight branching 的 C++ 接口和 python 接口是否会保持一致? TensorRT-LLM 安装成本高,今后是否有改进计划? TensorRT-LLM 会和 VLLM 发展角度有不同吗?

A5: 我们会尽量提供 c++ runtime 和 python runtime —致的接口,已经在规划当中。之前团队的重点在提升性能、完善功能上,以后在易用性方面也会不断改善。这里不好直接跟 vllm 的比较,但是 NVIDIA 会持续加大在 TensorRT-LLM 开发、社区和客户支持的投入,为业界提供最好的 LLM 推理方案。

以上就是本次分享的内容,谢谢大家。



发布于 2024-01-31 18:54 · IP 属地北京

内容所属专栏

