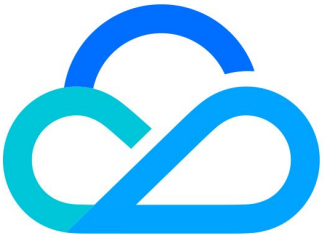
计算加速套件 TACO Kit

TACO LLM 推理加速引擎



腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

TACO LLM 推理加速引擎
TACO LLM 概述
产品简介

最近更新时间：2024-12-27 21:30:43

TACO-LLM（TencentCloud Accelerated Computing Optimization LLM）是基于腾讯云异构计算产品推出的一款大语言模型（LLM）推理加速引擎，用于提高大语言模型的推理效能。通过充分利用计算资源的并行计算能力，TACO-LLM 能够同时处理更多的大语言模型推理请求，为客户提供兼顾高吞吐和低时延的优化方案。TACO-LLM 可以降低生成结果的等待时间，提高推理流程效率，助您优化业务成本。

TACO-LLM 的优势

高易用性

TACO-LLM 设计实现了简单易用的接口，完全兼容业界开源 LLM 推理框架 vLLM。如果您正在使用 vLLM 作为推理引擎，可以无缝迁移到 TACO-LLM，轻松获得比 vLLM 更优的性能。此外，TACO-LLM 接口的简单易用性，使得使用其他推理框架的用户也能快速上手。

多计算平台支持

TACO-LLM 支持 GPU(Nvidia / AMD / Intel) / CPU(Intel / AMD) / TPU等多种计算平台，后续还会对主要国产计算平台进行支持。

高效能

TACO-LLM 使用 Continuous Batching / Paged Attention / 投机采样 /Auto Prefix Caching / CPU 辅助加速 / 长序列优化等多种 LLM 推理加速技术，针对不同的计算资源进行性能优化，全方位提升 LLM 推理计算的效能。

应用场景

最近更新时间：2025-04-01 10:58:22

TACO-LLM 适用于大语言模型的推理加速业务，可满足多种业务场景下推理提效的需求。以下是一些典型业务场景：

客户服务

业务场景	场景解释
智能客服	用于回答客户咨询，提供24小时服务。
问答系统	自动识别客户意图并作出响应。
情感分析	分析客户反馈和评论，识别客户情绪，帮助企业改进服务。

内容创作与编辑

业务场景	场景解释
文本润色	对用户提供的文本进行润色加工，生成质量更高的文本内容。
文本摘要	提取长篇文章的要点，生成摘要。
文本写作	生成新闻报道、文章、博客等内容。

翻译与本地化

业务场景	场景解释
机器翻译	将一种语言的文本翻译成另一种语言。
内容本地化	根据不同地区的文化和习惯调整内容。

编程开发

业务场景	场景解释
代码助手	根据自然语言描述或者已有代码片段自动生成代码，辅助开发者编程。
代码审查	自动检查代码中的错误和潜在问题。

教育培训

业务场景	场景解释
学习助手	根据用户的需求提供定制化的学习材料和解答。
RAG 知识引擎	结合外部知识库和大语言模型的能力，构建出强大的知识引擎。

LLM 训练辅助

业务场景	场景解释
预训练数据生成	利用大语言模型辅助生成大量预训练数据，供后续训练大语言模型使用。

TACO-LLM 在对时延敏感的在线服务场景(如智能客服和问答系统)和要求高吞吐的离线服务场景(如预训练数据生成和文本摘要)都提供了极具竞争力的性能加速方案，可帮助您最大效率地利用算力资源实现高吞吐，低延时，大幅降低单任务平均算力成本，获得最佳成本优化方案。

主要特性

最近更新时间：2025-04-17 15:08:12

LLM 服务部署的挑战

与传统 AI 场景不同，大语言模型服务关注多个维度的性能指标，它们最终对应不同层次的用户体验和服务质量。这些指标大体概括起来分为4个：

- **首字延迟（L1）**

  定义为 LLM 服务处理完 prompt 输出第一个 token 的延时，决定了用户从输入请求到获得响应的时间。对于实时的在线应用，低延迟很重要。但对偏离线的应用则该指标没有那么重要。该指标的好坏通常取决于推理引擎处理 prompt 并生成首字的时间。
- **解码延迟（L2）**

  定义为每个用户请求生成后续输出的平均响应时间，也是用户直观体验模型执行快慢的时间。假设执行速度是平均每 Token 100毫秒，则用户平均每秒可以得到10个 Tokens 的输出，每分钟 450 左右英文词。
- **请求延迟（L3）**

  定义为对给定用户产生完整响应的延迟。计算规则：L3 = L1 + L2 × 生成的 Token 数。
- **吞吐**

  定义为推理服务器面对全部用户和他们请求的流量时每秒可以生成的 Token 数量。
  部分推理引擎只关注或对上述某个指标有较好效果。而 TACO-LLM 均衡关注上述全部指标，并对各指标的实际部署效果均实现了全流程的优化。

LLM 部署的挑战来源于几个方面：

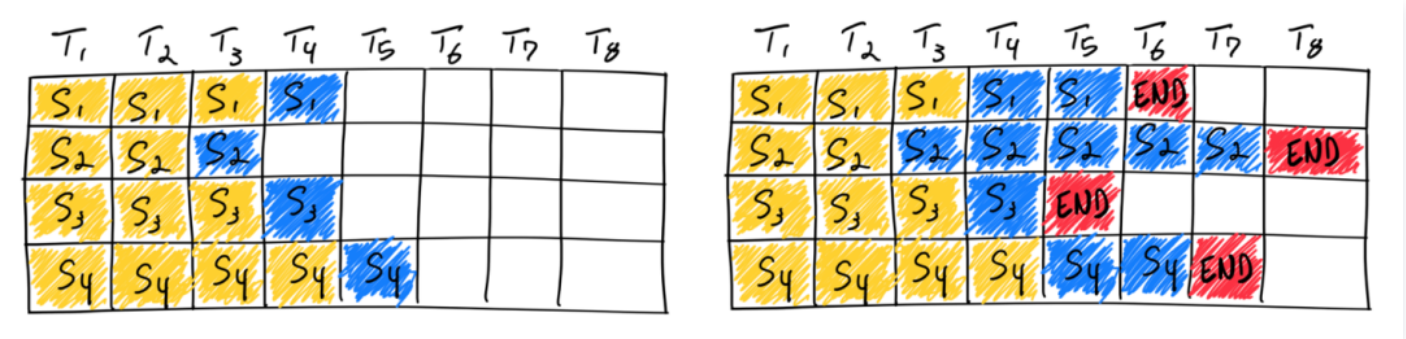
- 当前主流的 decoder-only 模型都具备自回归解码属性。模型生成输出是一个串行的计算过程。下一个输出依赖上一个输出。因此很难发挥出 GPU 或其他加速硬件的并行加速能力。同时，较低的 Arithmetic Intensity 对显存带宽的利用也提出了挑战。
- 大模型的**大**对显存容量提出了最直接的挑战。
- 传统的 Transformer 推理框架将 KV-Cache 按 batchsize 和 sequence length 维度组织数据，这会导致两个潜在的性能陷阱：
  - 请求的输出有长有短。这种数据组织方式需要等一个 Batch 中最长输出长度的请求计算完才能完成整个 Batch 的计算。在此之前，新的请求无法开始计算。而已计算完的请求，只能进行无效计算，消耗有效算力。同时，这种方式管理显存效率较低，无法做到“实用实销”，且随着不同请求计算过程中的显存使用，会造成显存碎片，进一步加剧资源瓶颈。
  - 目前很多优秀的 attention 加速技术实际上是按上述数据 layout 来实现的。例如 flash-attention、flash-decoding 等。这意味着更高效的重新设计，例如下文中提到的 Paged Attention 技术将无法直接享受到社区优化红利，仍给我们留出了进一步的优化空间。

为了有效应对上述挑战，腾讯云异构计算研发团队倾力打造了一款面向生产的 LLM 推理引擎。全方位应对上述挑战。

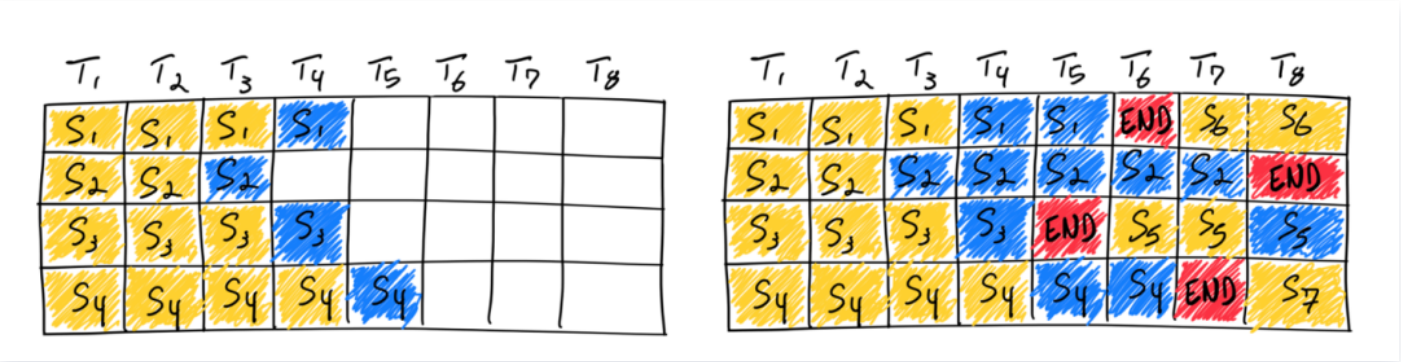
Continuous Batching

传统的 Batching 方式被称为 Static Batching。如上文所述，Static Batching 方式需要等一个Batch 中最长输出长度的请求完成计算，整个 Batch 才完成返回，新的请求才能重新 Batch 并开始计算。因此，Static Batching 方式在其他请求计算完成，等待最长输出请求计算的过程中，严重浪费了硬件算力。TACO-LLM 通过 Continuous Batching 的方式来解决这个问题。Continuous Batching 无需等待 Batch 中所有请求都完成计算，而是一旦有请求完成计算，即可以加入新的请求，实现迭代级别的调度，提高计算效率。从而实现较高的 GPU 计算利用率。

Static Batching



Continuous Batching

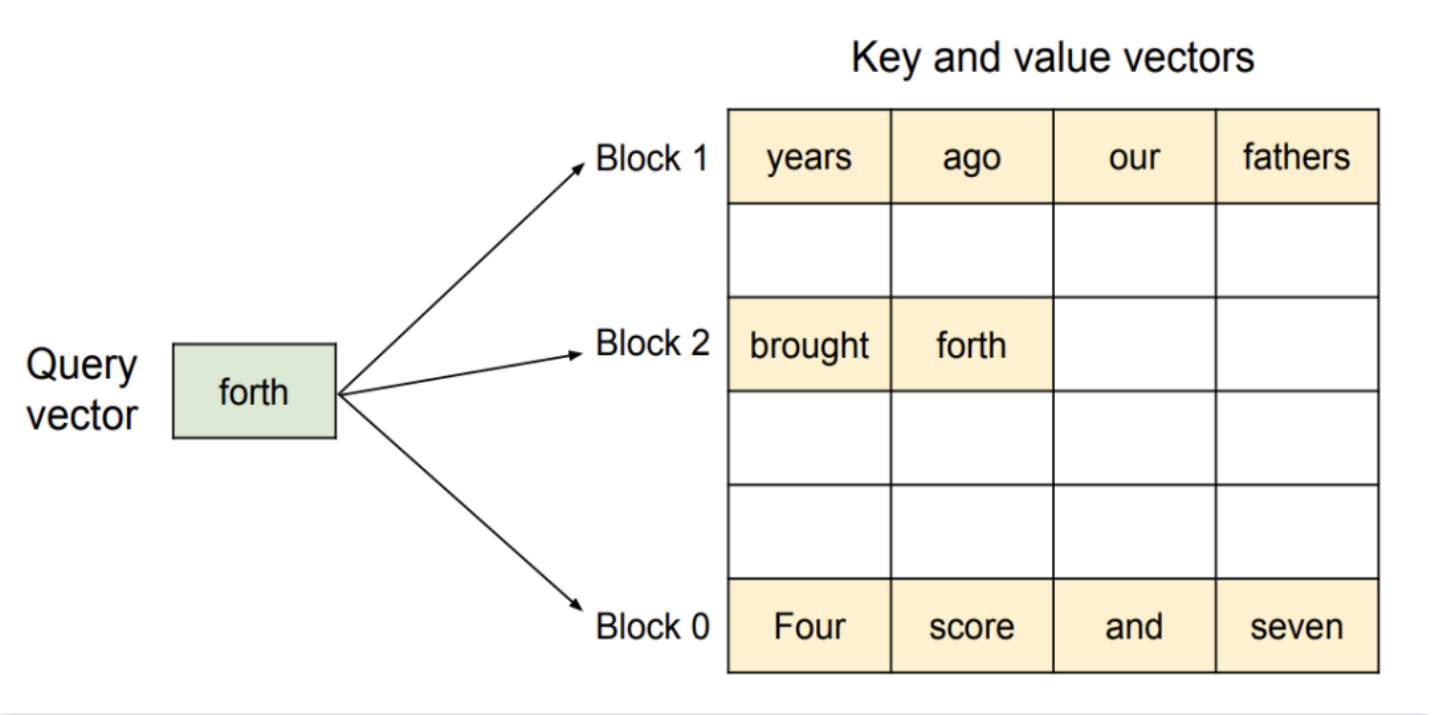


Paged Attention

大模型推理计算性能优化一个常用的方式是 KV-Cache 技术。Transformer 层的 attention 组件计算当前 Token value 值时，需要依赖之前 Token 序列的 Key 和 Value 值。KV-Cache 通过存储之前 Token 序列的 Key 和 Value 的值，避免后续计算中，重复计算 Key 和 Value 值，提高整体计算性能，是一种以空间换时间的优化策略。

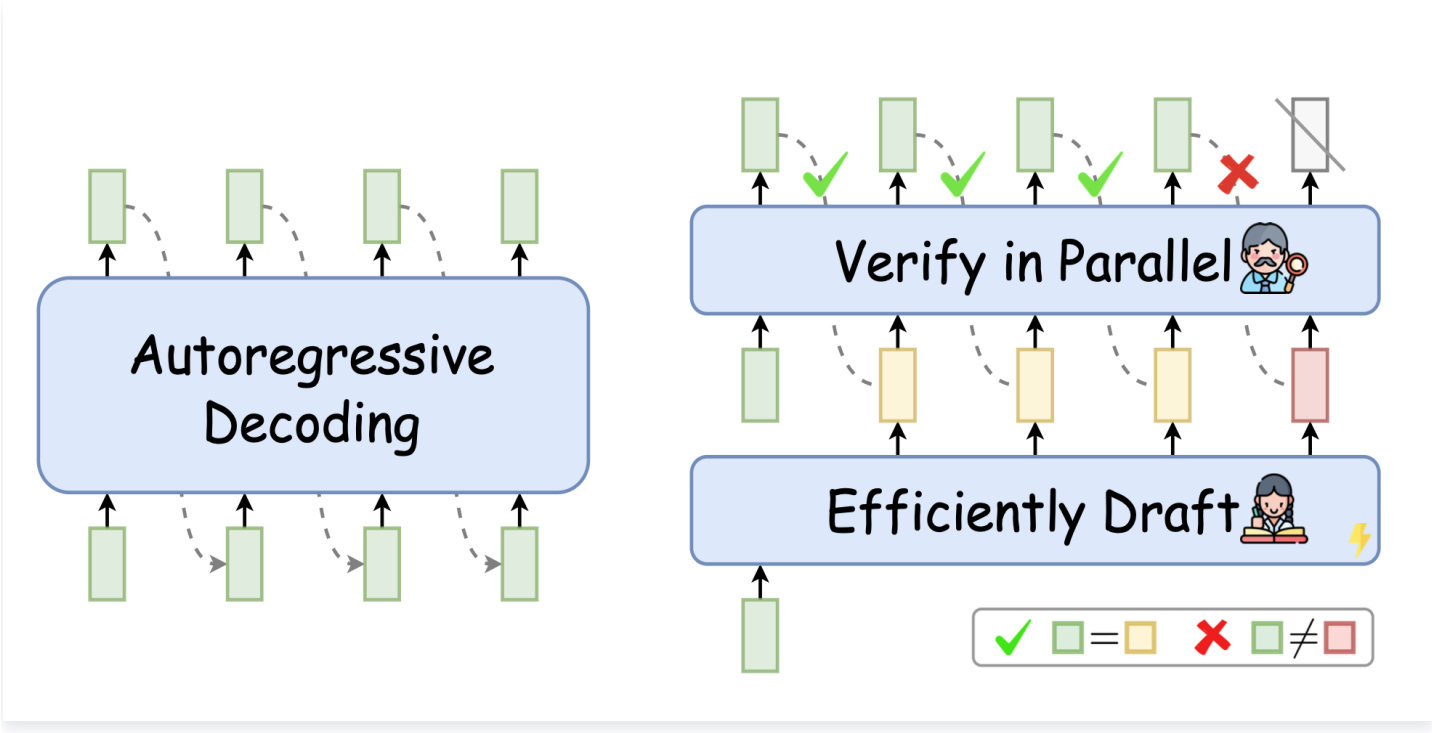
传统的 KV-Cache 实现机制是在显存中提前预留一块连续的存储空间来存储 Key 和 Value 值。但是，随着存储资源的分配和释放，显存中会存在很多“碎片”。某些情况下，虽然剩余的显存总量大于 KV-Cache 所需，但是由于不存在一块连续的存储空间可以满足 KV-Cache，计算也无法进行。

Paged Attention 是一种新的 KV-Cache 实现方式，它从传统操作系统的概念中获得灵感，例如分页和虚拟内存，允许 KV-Cache 通过分配固定大小的“页”或“块”在物理非连续内存上实现逻辑连续。然后可以将注意力机制重写为在块对齐的输入上运行，从而允许在非连续的内存范围内执行注意力计算。TACO-LLM 通过 Paged Attention 技术，实现了较高的显存利用效率。



投机采样

大语言模型的自回归解码属性要求每次生成新的 Token，都需要依赖所有已解码的 Token，且需要重新加载模型全部权重进行串行解码。这种计算方式无法充分利用 GPU 的算力，计算效率不高，解码成本高昂。而 TACO-LLM 通过投机采样的方式，从根本上解决了计算访存比的问题。通过引入一个高效的 Draft 辅助解码，可以让真正部署的大模型实现“并行”解码，从而大幅提高解码效率。我们称之为 **Spective Sampling（SpS）**技术。

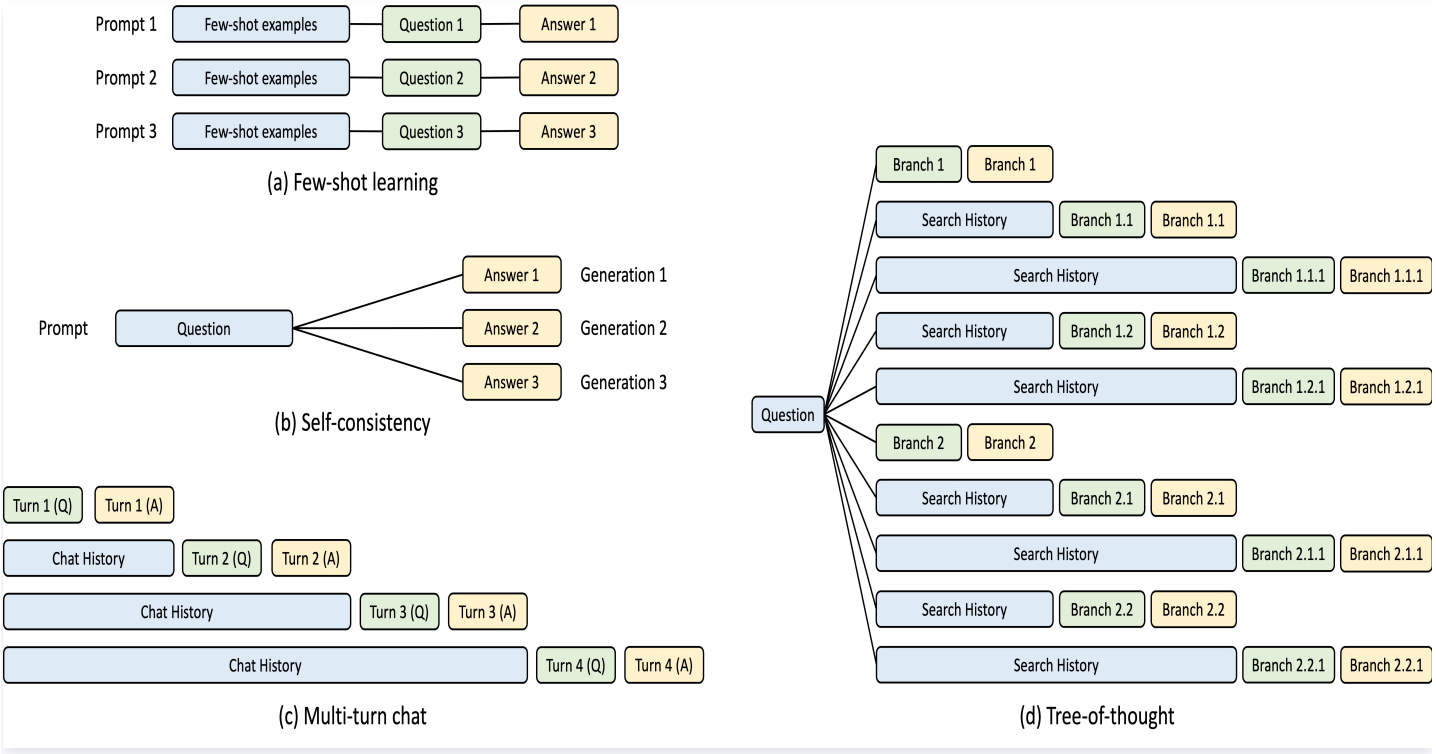


TACO-LLM 支持多种投机采样技术，如基于上下文的 Lookahead-Cache 和基于模型的 Eagle/Medusa 等，在多种业务场景中，均能实现高效的 LLM 推理计算。关于 Lookahead-Cache 的详情和使用方式请参见 Lookahead Cache 。

Auto Prefix Caching

LLM 推理计算主要分为两个过程：**Prefill 阶段（Prompt 计算）**和 **Decode 阶段**。这两个阶段的计算特性存在不同，Prefill 阶段是计算受限的，而 Decode 阶段是访存受限的。为了避免重复计算，Prefill 阶段主要作用就是给 Decode 阶段准备 KV Cache。但这些 KV Cache 通常只是为单条推理请求服务的，当请求结束，对应的 KV-Cache 就会清除。

KV Cache 能不能跨请求复用？在某些 LLM 业务场景下，多次请求的 Prompt 可能会共享同一个前缀（Prefix），比如少量样本学习，多轮对话等。在这些情况下，很多请求 Prompt 的前缀的 KV Cache 计算的结果是相同的，可以被缓存起来，给之后的请求复用。TACO-LLM 的 Auto Prefix Cache 技术可以针对这种场景进行优化，使得具有相同 Prompt 前缀的 KV-Cache 可以跨请求复用，降低计算开销，提升推理计算性能。详情和使用方式请参见 Auto Prefix Cache 。



量化

随着以 Transformer 为基石的大语言模型规模的快速增大，LLM 的推理部署对 GPU 显存和算力的需求激增。如何减少大语言模型部署的 GPU 显存需求，提高计算效率，降低推理部署成本就显得愈发重要。模型量化是解决这个问题非常重要的一种手段。业界提出了多种适用于LLM的量化算法，如GPTQ、AWQ、FP8 等，在保证模型精度损失满足业务需求的情况下，显著提升 LLM 推理计算的效能，降低部署成本。TACO-LLM 对业界主要的 LLM 量化算法均进行了适配支持，详情和使用方式请参见 量化 。

CPU 辅助加速

传统的投机采样使用 GPU 作为 Draft Model 的计算资源，而 GPU 的成本高昂。为了进一步降低计算成本，TACO 团队与 Intel 团队合作，基于 AMX 指令集对 CPU 上的矩阵乘法做了优化，使得使用 CPU 作为 Draft Model 成为可能，从而在进行推理加速的同时，显著降低了推理成本。详情和使用方式请参见 CPU 辅助加速 。

长序列并行

在 LLM 大模型推理中，长序列场景应用越来越广泛。目前业界对长序列的优化主要有 KV-Cache 量化、稀疏化等，这些都对模型精度有一定的影响。TACO LLM 自研长序列并行方案，可以在长序列场景进行精度无损的加速。在长序列推理场景，LLM 推理的首字延迟较高。针对该问题，序列并行在 Prefill 阶段采用了 Ring Attention 类的方案，可以通过扩展机器，降低首字延迟。而对于推理阶段，我们可以使用 Lookahead 等投机采样技术加速。详情和使用方式请参见 长序列优化 。

参考文献

[1] How continuous batching enables 23x throughput in LLM inference while reducing p50 latency
[2] Efficient Memory Management for Large Language Model Serving with PagedAttention
[3] Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding
[4] Fast and Expressive LLM Inference with RadixAttention and SGLang

支持模型

最近更新时间：2024-12-27 21:30:43

TACO-LLM 支持 Huggingface 模型格式的多种生成式 Transformer 模型。下面列出了 TACO-LLM 目前支持的模型架构和对应的常用模型。

Decoder-only 语言模型

Architecture	Models	Example HuggingFace Models	LoRA
BaiChuanForCausalLM	Baichuan & Baichuan2	baichuan-inc/Baichuan2-13B-Chat, baichuan-inc/Baichuan-7B, etc.	✓
BloomForCausalLM	BLOOM, BLOOMZ, BLOOMChat	bigscience/bloom, bigscience/bloomz, etc.	-
ChatGLMModel	ChatGLM	THUDM/chatglm2-6b, THUDM/chatglm3-6b, etc.	✓
FalconForCausalLM	Falcon	tiiuae/falcon-7b, tiiuae/falcon-40b, tiiuae/falcon-rw-7b, etc.	-
GemmaForCausalLM	Gemma	google/gemma-2b, google/gemma-7b, etc.	✓
Gemma2ForCausalLM	Gemma2	google/gemma-2-9b, google/gemma-2-27b, etc.	✓
GPT2LMHeadModel	GPT-2	gpt2, gpt2-xl, etc.	-
GPTBigCodeForCausalLM	StarCoder, SantaCoder, WizardCoder	bigcode/starcoder, bigcode/gpt_bigcode-santacoder, WizardLM/WizardCoder-15B-V1.0, etc.	✓
GPTJForCausalLM	GPT-J	EleutherAI/gpt-j-6b, nomic-ai/gpt4all-j, etc.	-
GPTNeoXForCausalLM	GPT-NeoX, Pythia, OpenAssistant, Dolly V2, StableLM	EleutherAI/gpt-neox-20b, EleutherAI/pythia-12b, OpenAssistant/oasst-sft-4-pythia-12b-epoch-3.5, databricks/dolly-	-

		v2-12b, stabilityai/stablelm-tuned-alpha-7b, etc.	
InternLMForCausalLM	InternLM	internlm/internlm-7b, internlm/internlm-chat-7b, etc.	✓
InternLM2ForCausalLM	InternLM2	internlm/internlm2-7b, internlm/internlm2-chat-7b, etc.	-
LlamaForCausalLM	Llama 3.1, Llama 3, Llama 2, LLaMA, Yi	meta-llama/Meta-Llama-3.1-405B-Instruct, meta-llama/Meta-Llama-3.1-70B, meta-llama/Meta-Llama-3-70B-Instruct, meta-llama/Llama-2-70b-hf, 01-ai/Yi-34B, etc.	✓
MistralForCausalLM	Mistral, Mistral-Instruct	mistralai/Mistral-7B-v0.1, mistralai/Mistral-7B-Instruct-v0.1, etc.	✓
MixtralForCausalLM	Mixtral-8x7B, Mixtral-8x7B-Instruct	mistralai/Mixtral-8x7B-v0.1, mistralai/Mixtral-8x7B-Instruct-v0.1, mistral-community/Mixtral-8x22B-v0.1, etc.	✓
NemotronForCausalLM	Nemotron-3, Nemotron-4, Minitron	nvidia/Minitron-8B-Base, mgoin/Nemotron-4-340B-Base-hf-FP8, etc.	✓
OPTForCausalLM	OPT, OPT-IML	facebook/opt-66b, facebook/opt-iml-max-30b, etc.	
PhiForCausalLM	Phi	microsoft/phi-1_5, microsoft/phi-2, etc.	✓
Phi3ForCausalLM	Phi-3	microsoft/Phi-3-mini-4k-instruct, microsoft/Phi-3-mini-128k-instruct, microsoft/Phi-3-medium-128k-instruct, etc.	-
Phi3SmallForCausalLM	Phi-3-Small	microsoft/Phi-3-small-8k-instruct, microsoft/Phi-3-small-128k-instruct, etc.	-
PhiMoEForCausalLM	Phi-3.5-MoE	microsoft/Phi-3.5-MoE-instruct , etc.	-

QWenLMHeadModel	Qwen	Qwen/Qwen-7B, Qwen/Qwen-7B-Chat, etc.	-
Qwen2ForCausalLM	Qwen2	Qwen/Qwen2-beta-7B, Qwen/Qwen2-beta-7B-Chat, etc.	✓
Qwen2MoeForCausalLM	Qwen2MoE	Qwen/Qwen1.5-MoE-A2.7B, Qwen/Qwen1.5-MoE-A2.7B-Chat, etc.	-
StableLmForCausalLM	StableLM	stabilityai/stablelm-3b-4e1t/ , stabilityai/stablelm-base-alpha-7b-v2, etc.	-
Starcoder2ForCausalLM	Starcoder2	bigcode/starcoder2-3b, bigcode/starcoder2-7b, bigcode/starcoder2-15b, etc.	-
XverseForCausalLM	Xverse	xverse/XVERSE-7B-Chat, xverse/XVERSE-13B-Chat, xverse/XVERSE-65B-Chat, etc.	-

多模态语言模型

Architecture	Models	Modalities	Example HuggingFace Models	LoRA
InternVLChatModel	InternVL2	Image(E+)	OpenGVLab/InternVL2-4B, OpenGVLab/InternVL2-8B, etc.	-
LlavaForConditionalGeneration	LLaVA-1.5	Image(E+)	llava-hf/llava-1.5-7b-hf, llava-hf/llava-1.5-13b-hf, etc.	-
LlavaNextForConditionalGeneration	LLaVA-NeXT	Image(E+)	llava-hf/llava-v1.6-mistral-7b-hf, llava-hf/llava-v1.6-vicuna-7b-hf, etc.	-
LlavaNextVideoForConditionalGeneration	LLaVA-NeXT-Video	Video	llava-hf/LLaVA-NeXT-Video-7B-hf, etc. (see note)	-
PaliGemmaForConditionalGeneration	PaliGemma	Image(E)	google/paligemma-3b-pt-224, google/paligemma-3b-mix-224, etc.	-

Phi3VForCausalLM	Phi-3-Vision, Phi-3.5-Vision	Image(E+)	microsoft/Phi-3-vision-128k-instruct, microsoft/Phi-3.5-vision-instruct etc.	-
PixtralForConditionalGeneration	Pixtral	Image(+)	mistralai/Pixtral-12B-2409	-
QWenLMHeadModel	Qwen-VL	Image(E+)	Qwen/Qwen-VL, Qwen/Qwen-VL-Chat, etc.	-
Qwen2VLForConditionalGeneration	Qwen2-VL (see note)	Image(+) / Video(+)	Qwen/Qwen2-VL-2B-Instruct, Qwen/Qwen2-VL-7B-Instruct, Qwen/Qwen2-VL-72B-Instruct, etc.	-

说明：

- **E: 表示 Pre-computed embeddings 可以作为多模态输入。**
- **+: 表示一个 prompt 可以插入多个多模态输入。**

TACO-LLM 部署

最近更新时间：2025-01-14 14:51:12

TACO-license 部署

TACO-license 是 TACO-LLM 项目中用于鉴权和记录日志的组件。

公有云及太极客户：

- 公有云目前支持以下地域：南京、北京、上海、广州。
- 太极目前对所有客户提供支持。
- TACO-license 已经在上述范围生效，客户无需关注 TACO-license 部署。
  不在上述范围的公有云客户请进入官方 TACO 团队交流群 咨询使用特殊版本。

TCE/TCS 客户：

如需在 TCE/TCS 上部署 TACO-license，**请进入官方 TACO 团队交流群** 团队进行对接。

私有化客户：

TACO-license 背景及适用范围

TACO-license 是 TACO-LLM 项目下用于鉴权和记录日志的组件，适用于私有化部署 TACO-LLM。私有化客户需要在集群中部署 server 端，以对 client 端（即 TACO-LLM 本身）的请求进行鉴权。搭建条件如下，本文档中将使用 $your_license_version 来代表实际的版本号，请在代码中进行相应替换：

```
//搭建环境：需要K8s，server以service的形式部署并提供服务
docker images: taco-license-server:$your_license_version
```

成功鉴权后，可以正常使用 TACO-LLM 功能。

签发 license

只有被授权并在过期时间之前，在最大同时访问数量（如果设定）的限制下，才能成功鉴权的 GPU 才能使用。最大同时访问数量指的是同时访问 TACO-license 的 GPU 的最大数量，例如：授权10000台 GPU，同时限制最大同时访问的 GPU 数量为10台。

1. 客户将需要签发 license 的 GPU 发送给 TACO 团队，其方法为在所有可能执行 GPU 的机器上执行：

   ```
   nvidia-smi -L
   ```

   将结果写入文件（例如 request.txt），多台机器的结果直接按顺序粘贴合并到一个文件中，并提供过期时间以及最大访问数量，交给 TACO 团队以签发许可证。

2. 签发的许可证文件名为 license-taco_xxxxxxx.dat（例如 license-taco_1234-leolingli-20240906104354.dat）。

搭建 taco-license-server

1. 准备好 taco-license-server 的镜像，可以下载到本地或者上传到可供下载的位置。此包由 TACO 团队提供，下载地址为：

```
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/taco-llm/license-server/latest/taco-license-server-latest.tgz

#将docker image 解压或放到集群可以下载到的地址，如：
docker load -i taco-license-server-latest.tgz
#docker pull or  docker tag taco-license-server: $your_license_version \#$your_license_repository/taco-license-server:$your_license_version
```

2. 使用 Helm Chart 搭建服务，可以参考以下示例：

```
#install helm first
#wget taco-operator-idc package
wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/taco-llm/license-server/latest/taco-operator-idc-latest.tgz

#假设您的taco-license-server地址为 $your_license_repository/taco-license-server:$your_license_version
helm install --generate-name \
    --set global.repository="$your_license_repository" \

./taco-operator-idc-latest.tgz
```

3. 服务将部署在 kube-system 命名空间中，可以通过 curl 命令来验证是否成功搭建，截图如下：

```
[root@vm-1-143-centos ~]# k get svc
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)     AGE
kubernetes            ClusterIP   10.10.0.1      <none>        443/TCP     87d
taco-license-server   ClusterIP   10.10.195.83   <none>        10080/TCP   20h
[root@vm-1-143-centos ~]# curl 10.10.195.83:10080
taco license server
[root@vm-1-143-centos ~]#
```

导入 license

将上述签发的 license（license-taco_xxxxxxx.dat）导入集群，使 license 生效：

1. 在集群中执行以下操作以导入许可证。请注意，导入操作将覆盖所有现有的许可证。因此，新签发的许可证需要替代之前的所有许可证：

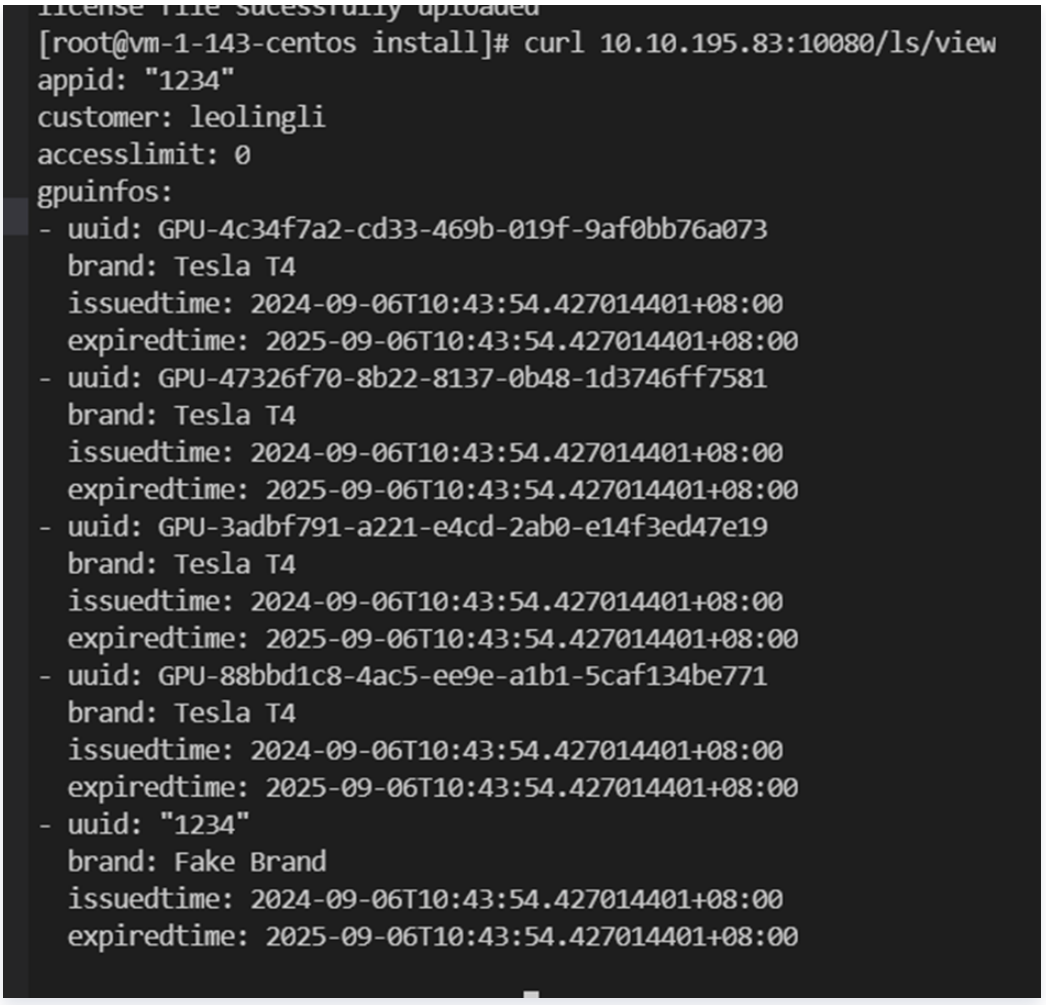
```
LICENSE_FILE_NAME=license-taco_xxxxxxxxxxx.dat

LICENSE_SERVER_IP=$(kubectl get svc -n kube-system taco-license-
server  -o jsonpath="{.spec.clusterIP}")

curl -X POST -F "file=@$LICENSE_FILE_NAME"
http://${LICENSE_SERVER_IP}:10080/ls
```

2. 执行上述命令后，可以看到提示，显示 license 已成功导入。

```
[root@vm-1-143-centos install]# curl -X POST -F "file=@$LICENSE_FILE_NAME" http://${LICENSE_SERVER_IP}:10080/ls
license file sucessfully uploaded
[root@vm-1-143-centos install]#
```

您也可以通过以下方法查看签发的许可证信息：

```
curl your-service-ip:10080/ls/view
```



client 端使用

1. 在容器中，只需使用 taco-llm 即可，无需特殊配置。您可以尝试在 pod 中执行以下操作：

```
curl http://taco-license-server.kube-system.svc.cluster.local:10080
```

2. 可以看到相同的输出：

```
taco license server
```

这意味着客户端可以访问许可证。

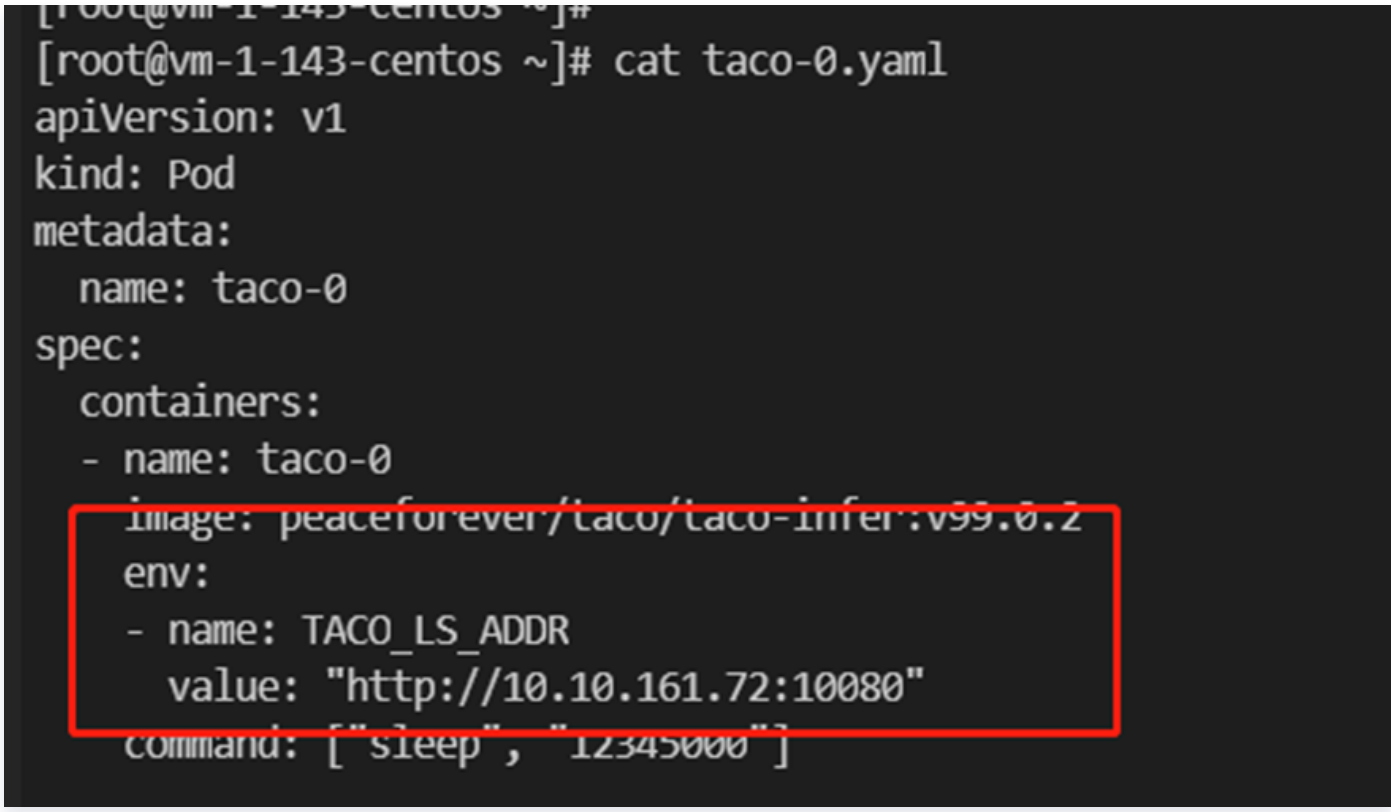
特殊情况：自定义 license 地址

> **说明：**
> 本章节适用于使用非 kube-system 命名空间或希望自定义 qgpu-license-server 部署方式的用户。

Taco-license-server 默认支持两种连接方式，即通过 http://taco-license-server.kube-system.svc.cluster.local:10080 和 http://taco-license-server:10080 访问。这意味着部署在 kube-

system 命名空间和与负载 pod 处于同一命名空间的服务地址都可以进行访问。如果用户想在其他命名空间使用，或者由于配置原因无法公开 taco-license-server 地址以允许客户端鉴权，则需要用户自行解决 taco-license-server 的部署问题。这包括部署一个可以访问到 taco-license-server 地址的 pod，并且设置 **TACO_LS_ADDR** 环境变量为 taco-license-server 的地址。
例如，如果 taco-license-server 可以通过地址 http://10.10.161.72:10080 访问，则应在 pod 的环境变量中添加以下变量：

- 可以通过配置 YAML 文件中的环境变量来实现。



- 或在容器中 bashrc 文件中添加变量：

```
export TACO_LS_ADDR=http://$your_license_addr:10080
```

Taco-LLM 将尝试通过此地址访问 Taco-License-Server。

> ⚠ **注意：**
> 如果要显式设置 TACO_LS_ADDR 的值，请确保将 TACO_LS_ADDR 的值导出为正确的 taco-license-server 地址。尝试访问无效地址会导致鉴权程序等待超时（通常为30秒），从而延迟鉴权启动时间，可能会影响到最初几次鉴权的结果。

TACO LLM 安装

最近更新时间：2024-12-27 21:30:43

环境准备

TACO-LLM 需要依赖 GPU 相关的基础软件，如 GPU 驱动 /CUDA 等。为了避免基础软件依赖导致 TACO-LLM 无法正常运行，我们提供了 TACO-LLM docker环境镜像，建议您优先使用该镜像作为 TACO-LLM 的运行环境。按照如下命令可以获取 docker 镜像并启动容器环境：

```
docker run -it \
    --privileged \
    --net=host \
    --ipc=host \
    --shm-size=16g \
    --name=taco_llm \
    --gpus all \
    -v /home/workspace:/home/workspace \
    ccr.ccs.tencentyun.com/taco/tacollm-dev:latest /bin/bash
```

安装 whl 包

> **说明：**
> 如果您有任何业务需求需要试用 TACO-LLM，请通过在线售后联系 TACO 团队。

1. 通过在线售后获取 TACO-LLM whl 安装包之后，可以按照如下命令在容器环境中安装 TACO-LLM：

```
pip3 install taco_llm-${version}-cp310-cp310-linux_x86_64.whl
```

2. 安装 TACO-LLM whl 包时，会自动安装相关的 python 依赖包。

TACO LLM 使用
离线模式

最近更新时间：2024-12-27 21:30:43

在离线推理业务场景中，您可以通过离线模式使用 TACO-LLM。本文档通过一个简单的例子介绍了如何使用 TACO-LLM 的离线模式。

导入 LLM 和 SamplingParams

首先，需要从 taco_llm 导入所需使用的 LLM 和 SamplingParams 类：

```
from taco_llm import LLM, SamplingParams
```

构建 prompts 和采样参数

接下来，构建所需的 prompts 和采样参数。本示例构建了4条 prompt，并设置了采样参数，其中 temperature 为0.8，top_p 为0.95。完整的采样参数配置可以参见 采样参数 API 。

```
# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)
```

构建 LLM 对象

接下来，我们将构建 LLM 实例。本示例使用了 facebook/opt-125m 模型以及其他默认配置参数来构建 LLM 实例。完整的 LLM 构建参数配置可以参见 离线API 。

```
# Create an LLM.
llm = LLM(model="facebook/opt-125m")
```

推理计算

最后，调用 LLM 对象的 generate 接口进行推理计算：

```
# Generate texts from the prompts. The output is a list of RequestOutput
objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

至此，TACO-LLM 的离线模式使用已经完成。以下是本示例的完整代码：

```
from taco_llm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts. The output is a list of RequestOutput
objects
# that contain the prompt, generated text, and other information.
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

在线模式

最近更新时间：2025-04-17 15:08:12

TACO-LLM 提供了实现 OpenAI Completions 和 Chat API 的 HTTP 服务端，您可以按照以下流程进行使用。

启动服务

首先，执行以下命令启动服务：

```
taco_llm serve facebook/opt-125m --api-key taco-llm-test
```

发送请求

您可以使用 OpenAI 的官方 Python 客户端来发送请求：

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:8000/v1",
    api_key="taco-llm-test",
)

completion = client.chat.completions.create(
    model="facebook/opt-125m",
    messages=[
        {"role": "user", "content": "Hello!"}
    ]
)

print(completion.choices[0].message)
```

您也可以直接使用 HTTP 客户端来发送请求：

```
import requests

api_key = "taco-llm-test"

headers = {
    "Authorization": f"Bearer {api_key}"
}
```

```
pload = {
    "prompt": "Hello!",
    "stream": True,
    "max_tokens": 128,
}

response = requests.post("http://localhost:8000/v1/completions",
                         headers=headers,
                         json=pload,
                         stream=True)

for chunk in response.iter_lines(chunk_size=8192,
                                 decode_unicode=False,
                                 delimiter=b"\0"):
    if chunk:
        data = json.loads(chunk.decode("utf-8"))
        output = data["text"][0]
        print(output)
```

完整服务端参数配置

执行 `taco_llm serve -h` 命令可以查看 TACO-LLM 完整的在线模式参数配置，详细内容请参见：在线模式 API 。

完整客户端参数配置

除了少部分参数不支持外，TACO-LLM 完全支持 OpenAI 的参数配置。您可以参见 OpenAI API 官方文档 查看完整的 API 参数配置。不支持的少部分参数配置如下：

- Chat: tools, and tool_choice。
- Completions: suffix。

基础配置

最近更新时间：2025-04-17 15:08:12

基本配置

以一个实际运行的例子进行离线测试为例：此例子可以参考离线模型的编写方式。

```
python3 offline_test.py \
    --model /models/Llama-2-7b-chat-hf/ \
    --tokenizer /models/Llama-2-7b-chat-hf/ \
    --dataset /datasets/ShareGPT_V3_unfiltered_cleaned_split.json \
    --num-prompts 16 \
    --max-num-batched-tokens 10240 \
    --max-num-seqs 32 \
    --trust-remote-code
```

参数说明：

- **--model**：指定模型的存放路径，支持相对路径、绝对路径或者 repo_id 则会从 huggingface 上下载，建议使用本地路径可加快运行速度。
- **--tokenizer**：同模型同一个参数。
- **--dataset**：指定当前测试 case 需要从哪个数据集采集 prompt 数据。
- **--num-prompts**：表示请求数。
- **--max-num-batched-tokens**：表示每次执行推理时支持最长的处理 token 数，多个 batch 的总和数，如果每个请求长，则会分多批完成请求。
- **--max-num-seqs**：后端支持最大的 batch 数，配置值建议参考实际 GPU 总显存及处理请求的长短值考虑。如果配置过大启动服务会慢，配置过小可能影响最终的吞吐性能。
- **--trust-remote-code**：表示信任当前部署，如果使用 model repo_id 时不加此参数可能导致模型加载失败。

除了上面参数外，还有一些可选配置参数供参考。

多卡推理

```
--tensor-parallel-size 1 \
```

--tensor-parallel-size： tensor 并行，支持1，2，4，8，会被模型的 layer 层数整除即可。

cudagraph 优化

```
--enforce-eager \
```

--enforce-eager：默认支持 CUDA Graph 优化。如果不想使用 CUDA Graph，可以添加 --enforce-eager 参数。默认配置的 CUDA Graph 在启动模型时会捕获图的时间开销，max-num-seqs 配置越大，所需时间越长。

显存占用配置

```
--gpu-memory-utilization 0.9 \
--conservative-dry-run \
```

- **--gpu-memory-utilization**： 小数值取值范围：0.1 ~ 0.95 表示 GPU 卡占用显存，这部分显存主要有三部分：进程初始化显存（非 torch）、权重、kvcache；剩下的显存一般用于非 torch 分配、cuda graph 等场景，如果此值配置过高，max-num-seqs 配置过大，会导致 cuda graph 显存不够，导致初始化失败。
- **--conservative-dry-run**：增加此参数表示在量化等场景由于内部 kernel 也使用显存，而这部分显存并未统计到上述利用值上面，导致 OOM，建议在量化场景下增加此参数保证安全运行。

speculative推理

```
--speculative-model /models/*** --num-speculative-tokens 3 \
```

- **--speculative-model**： 后面跟 speculative 模型的路径，运行绝对路径、相对路径等。
- **--num-speculative-tokens**：3 表示 speculative 模型一次生成的 token 数，可以自行配置调优。

多步推理

```
--num-scheduler-steps 4 \
```

--num-scheduler-steps： 支持多步处理，可以配置，[1-8]等整数值，在 decoding 阶段，部分 cpu 结果的处理可以 overlap 到 GPU 上，增加吞吐值，默认不打开。

Lookahead Cache

最近更新时间：2025-08-04 10:53:21

默认方式开启

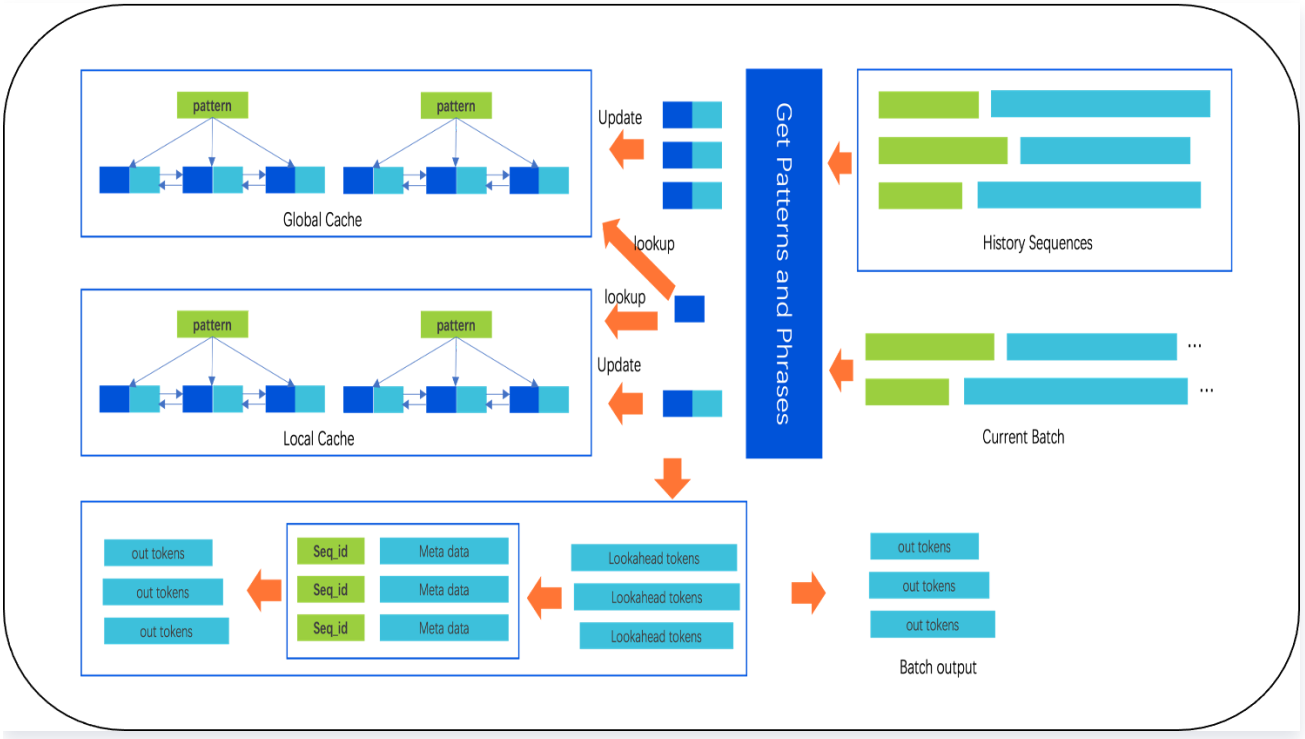
您只需在启动命令行中加入以下命令，即可开启 lookahead-cache：

```
--lookahead-cache-config-dir ./   # 或者任意其他目录
```

如果您只想了解 LookaheadCache 在默认配置下的加速效果（默认配置通常已足够优秀），那么以上便是您需要知道的所有信息。如果您希望进一步调整性能，选择不同的 CacheMode，或查看 debug log 等，那么您需要继续阅读后续内容。

基本原理

一句话概括 Lookahead Cache 的基本原理：使用历史的 Token 对（key: value）来预测当前 Token 对的 value 值。所提及的 Lookahead Cache 是 Taco-LLM完全自主研发的 Lookahead 技术，主要设计并实现了两个基础方案。下图展示了第一个方案的 Lookahead Cache：



- 方案一的 Lookahead Cache 参见了LLMA 和 N-gram 的设计思想，并在此基础上进一步优化，并增加了新特性。

  主要特点是：
  - 支持变长的 Lookahead Len；
  - 支持 batch 化；

- 支持变长 cuda-graph。

- 方案二抛弃了 N-gram 和 LLMA 的设计思想，重新设计，相比方案一继承了其所有优点，同时拥有更加高的平均命中长度，但是也更加复杂，这里不详细叙述。

  在功能上，方案二新增支持：

  - Multi-path 的多候选结果输出；
  - 基于前缀树的多候选结果的输出；
  - 树的动态裁剪；整体上的预测结果命中率是方案一的1.4x，部分场景是方案一的2.0x。

  但是相比方案一，方案二的冷启动问题更加明显，所以往往结合方案一一起使用。

进阶使用方式

配置介绍

在这种使用方式下，您需要提供一个命名为 `lookahead_cache_config.json` 的配置文件，该文件必须位于通过 --lookahead-cache-config-dir 指定的目录中。（文件名： lookahead_cache_config.json ）

```
{
    "cache_mode": 2,
    "cache_size": 5000000,
    "copy_length": 7,
    "match_length": 2,
    "turbo_match_length": 7,
    "min_match_length": 2,
    "cell_max_size": 16,
    "voc_size": 200000,
    "max_seq_len": 32768,
    "eos_token_id": 2,
    "top_k": 1,
    "threshold": 2.0,
    "decay": 0.5,
    "is_hybrid": true,
    "is_debug": true,
    "log_interval": 3000,
    "target_parallelism": 512,
    "top_k_in_cell": 16,
    "token_paths_top_k": 2,
    "start_freq": 10.0,
    "num_threads": 8,
    "global_cache_switch": true
}
```

还有选择性的 warmup_file 配置。

参数	参数解释
cache_mode	0表示 RawLookaheadCache（第一版），1 表示 TurboLookaheadCache（第二版），2表示两者混合的方式，混合方式：在 TurboLookaheadCache 返回结果为空时，使用 RawLookaheadCache 来预测，RawLookahead 的命中率更低，但是在开始阶段的触发概率更高，而 TurboLookaheadCache 的 warmup 的时间稍长，在测试数据很少的时候，建议 cache_mode设置为2，如果有充足的测试数据，建议设置 cache_mode 为1，因为其命中率和准确率在大部分场景下都更好，有更少的冗余计算。
cache_size	cache 的大小，可以不用设置，而通过 `taco-llm` 的命令行参数 --cpu-decoding-memory-utilization 来设置。它是一个0到1的值，表示使用当前环境下内存的比值，默认是0.15。
copy_length	lookahead 往前看的长度。
match_length	触发 lookahead cache 的匹配长度。
turbo_match_length	TurboLookaheadCache 的最大匹配长度。
min_match_length	TurboLookaheadCache 的最小匹配长度。
cell_max_size	二级缓存的大小，同一个匹配下不同词组，按照出现频率和时间，组成一个 LRU的 cache。
voc_size	tokenizer 中词表的大小。
max_seq_len	模型的最大序列长度，不用配置，直接可以从模型的 config 中获取。
eos_token_id	结束 token，不用配置，直接从模型的 config 文件中获取。
top_k	取二级 cache 频率最大，时间最近的词的数目。
threshold	二级 cache 满了以后，需要删除时，要删除对象的频率阈值。
decay	对于二级 cache 满了时，且所有的频率都高于 threshold 时，需要先衰减。
is_hybrid	针对 RawLookaheadCache 打开，多种 match-length 会结合，例如，match-length=3时，会混合 match-length=3，2，1。
is_debug	打开时，会输出 cache log。
log_interval	打印 log 的频次。

target_parallelism	最大并行度，即 sum(seq_lens)，开启了这个就会，在 sum(seq_lens) > target_parallelism 后会根据各个 seq 的命中情况来惩罚 copy_length。仅仅对 cache_mode=0 时有效。
top_k_in_cell	仅针对 TurboLookaheadCache，表示查找二级缓存时，一次返回 token 的数目。
token_paths_top_k	表示 beam search 的宽度，默认是1，在命中率比较低时，可以尝试把这个参数改为2，或者更大，建议在4以内。这个参数的开启在一定程度上增加了命中率，同时也会增加冗余计算。
start_freq	表示 local cache 的初始比重，越高表示优先保留 Local cache 中得到的 path，仅在 token_paths_top_k>1 时生效。
num_threads	表示在使用 TurboLookaheadCache 的并发度，提升的是 TurboLookaheadCache 本身的运行速度。
global_cache_switch	为 True 表示开启 global_cache，false 表示关闭，这样样本之间将不受影响。
ignore_prompt	忽略 prompt，多针对翻译等 prompt 和 generation tokens 毫无关系的场景。

注意的问题

ignore_eos

正确操作：ignore_eos=False
在测试 taco_llm lookahead 时，这个参数需要设置为 False，因为 LookaheadCache 里，遇到 eos 的 token_id 时，会把 local_cache 清除掉。将会影响hit_rate 和 global_average_hit_len 等指标，最终影响整体的性能。如果需要测试固定的输出长度，建议在搜索样本时，选择更长的输出样本，然后把output_len 固定，这样可以减少 output tokens 未到目标输出长度时，就遇到 eos token。

常用性能调节方式

> ⚠ **注意：**
> 以下的配置方式可以叠加使用。

小数据集测试

如果发现使用少量样本测试，TurboLookaheadCache（cache_mode=1）效果比 RawLookaheadCache（cache_mode=0）的性能还要差，此时建议用cache_mode=2的混合模式。增加如下配置：

```
{
    "cache_mode": 2
}
```

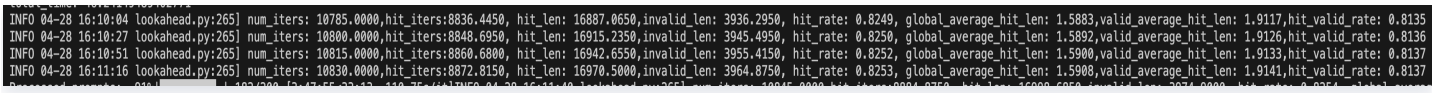
这是由于 TurboLookaheadCache 的冷启动问题导致的。

性能不符合预期

现象：使用 lookahead-cache 默认配置会有1.7x-3.x+的性能收益，如果没有达到这个收益。首先需要查看是否使用了 greedy 的采样方式（这样会有最好的性能）。如果因为业务的原因不能使用 greedy，可以在满足业务需要的情况下调小 temprature，尽可能的减少随机性。这样调节和 lookahead cache 的原理密切相关。lookahead cache 会将历史输入和输出的 tokens 都存放到 cache 中，如果输出的随机性太强，那么历史输入对当前输出的参考性，就会变弱，从而导致 cache 的命中率下降，加速性能也下降。
在上述配置都没有问题的情况下，依然不符合加速预期，可以在 lookahead cache 的配置目录下，增加含有如下内容的配置文件（文件名： lookahead_cache_config.json）：

```
{
    "is_debug": true,
    "log_interval": 300
}
```

添加此配置后，系统将输出 hit-rate 的相关指标，如下图所示。



各命中率指标的含义

- **num_iters**：平均每一个请求的 generation 阶段的自回归次数。
- **hit_iters**：平均每一个请求的命中长度大于0的迭代次数。
- **hit_len**：平均每一个请求的命中总长度。
- **invalid_len**: 平均每一个请求 lookahead 的总长度减去命中的总长度: lookahead_len - hit_len。
- **hit_rate**： 平均每一个请求命中的迭代次数除以总自回归次数: hit_iters / num_iters。
- **global_average_hit_len**：平均每一个请求的平均每次迭代的命中长度: hit_len / num_iters。
- **valid_average_hit_len**：平均每一个请求命中长度大于0的情况下，平均命中长度：hit_len / hit_iters。
- **hit_valid_rate**：平均每一个请求在总的命中长度除以总的 lookahead 长度：hit_len/ lookahead_len。

其中 **global_average_hit_len** 加1，即表示开启 lookahead 下，每一次 decoding 迭代平均吐出的 token 数目，理想情况下，这个数据表示 decoding 过程的加速倍数。例如上图所示，global_average_hit_len =

1.59，加1为2.59，那么理想情况下 decoding 阶段加速2.59倍。**hit_rate** 表示至少有一个命中的命中次数与全局迭代次数的比值。**valid_average_hit_len** 表示如果至少有一个 token 命中的情况下。平均的命中长度，这个数据加1，可以用来调节 copy_length 的大小，**注意**：如果使用了 mutli-path 优化，不能使用这个方式来调节。下面举例说明在采样方式没有问题时，如何调节 Lookahead Cache。

指标 global_average_hit_len<0.8

当 global_average_hit_len < 0.8 时，属于命中长度较低的情况。首先，确认是否开启了 MultiPath 功能。如果使用的是默认配置，即没有指定配置文件或者 cache_mode 项未进行配置，那么可以确定 MultiPath 已开启，且其值为 2。如果未开启，可添加以下配置：

```
{
    "is_debug": true,
    "log_interval": 3000,
    "token_paths_top_k": 2,
    "start_freq": 10.0
}
```

- 添加该配置后，会禁用并行度惩罚。因此，在大批量处理（>=64）时，冗余计算可能会显著增加，这可能会影响性能。如果性能下降是由此原因引起的，可以尝试降低 copy_length 的值。如果这些操作都无效，请检查 ignore_eos 是否设置为 False（如上文所述）。
- 确定 MultiPath 生效，但是 global_average_hit_len 依然较低，可以尝试增加路径的数量，即调整 token_paths_top_k 的设置。token_paths_top_k 的设定可以采用 copy_length / valid_average_hit_len 的方法，其中默认的 copy_length 为7。
- 如果此时 global_average_hit_len 指标仍然不高（<0.8），则需要检查测试场景，例如生成序列长度很短（<=32），或者输入和输出为语音、多模态等毫无关联的场景。在这种情况下，Lookahead Cache 在原理上受限，不会有很好的效果。

指标 global_average_hit_len>=0.6 && <=1 性能差

global_average_hit_len 在 [0.6, 1] 这个区间内，但是端到端的速度甚至没有提升，可以尝试以下操作：

- 大 batchsize(>=32)，此时冗余计算会比较多，lookahead 带来的收益可能不足以抵消冗余计算带来的开销，可以试着调低 copy_length，可以4，5，6分别尝试，如果测试数据足够，也是尝试将 cache_mode 修改为1，将 copy_length 调节到2，3等。
- 低算力卡(HCCPNV6, PNV6, PNV5b): 分析和调节方法与大 batchsize 场景类似，此时该问题可能会来的更早，例如 bs=16左右。

不同的样本请求差别较大时

不同的样本之间差别较大时，需要快速的更新 cache，增加时间局部性，可以尝试增加下面的一组调节参数。也可以通过 **cell_max_size** 指标的调节来改变cache 的变化速度，其值越小变化的越快，默认值是16，建议的调节范围是[8, 32].

```
{
    "start_freq": 1.0,
    "decay": 0.1,
    "cell_max_size": 16
}
```

warmup

文件格式为 JSON 格式，数据格式为 [{"prompt": [token_ids], "output": [token_ids]}]。处理后，将其路径设置到 lookahead_cache_config.json 中，字段名为 "warmup_file"：warmup_file_path。

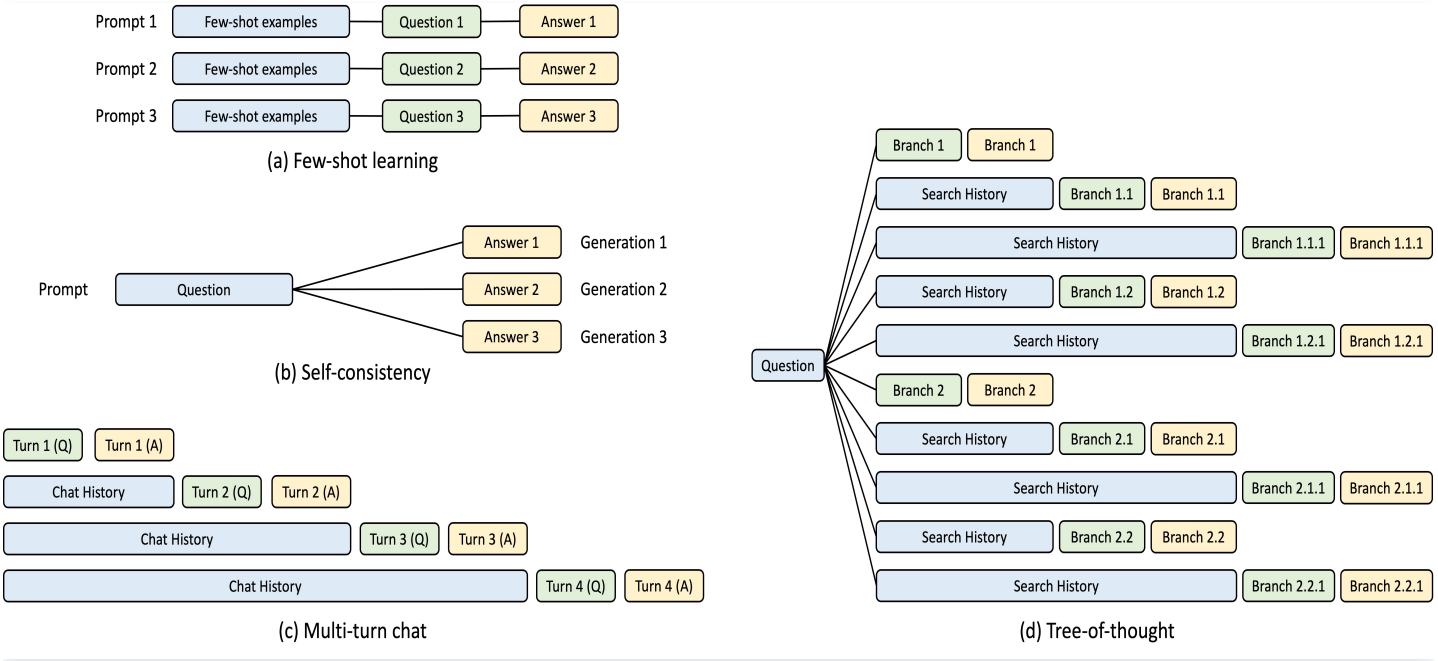
Auto Prefix Caching

最近更新时间：2025-04-17 15:08:12

在多轮对话、系统提示以及其他具有大量共同前缀的应用场景中，自动前缀缓存功能能够存储之前的前缀键值缓存，加快后续具有相同前缀请求的预填充阶段，从而降低首次响应时间，优化处理能力。

原理

LLM 推理计算主要分为两个过程：**Prefill 阶段（Prompt 计算）**和 **Decode 阶段**。这两个阶段的计算特性存在不同，Prefill 阶段是计算受限的，而 Decode 阶段是访存受限的。为了避免重复计算，Prefill 阶段主要作用就是给 Decode 阶段准备 KV Cache。但这些 KV Cache 通常只是为单条推理请求服务的，当请求结束，对应的 KV-Cache 就会清除。那很自然的一种想法就是，KV Cache 能不能跨请求复用？在某些 LLM 业务场景下，多次请求的 Prompt 可能会共享同一个前缀（Prefix），比如少量样本学习，多轮对话等。在这些情况下，很多请求 Prompt 的前缀的 KV Cache 计算的结果是相同的，可以被缓存起来，给之后的请求复用。TACO-LLM 的 Auto Prefix Cache 技术可以针对这种场景进行优化，使得具有相同 Prompt 前缀的 KV-Cache 可以跨请求复用，降低计算开销，提升推理计算性能。



具体详情请参见：Fast and Expressive LLM Inference with RadixAttention and SGLang 。

启动选项

```
--enable-prefix-caching
                        Enables automatic prefix caching.
```

在 server 启动指令中添加该指令即可开启 Auto Prefix Caching 功能。

Prefix Cache Offload

显卡的显存有限，除装载模型权重和运行激活空间以外，留给 prefix kv cache 的 blocks 数量是固定而有限的。当不同的 prefix 请求较多，随着请求的不断输入，之前的 prefix cache 就会被驱逐，之后有相同 prefix 的请求将无法命中 prefix cache。这导致需要重新执行 prefill 流程，从而无法获得加速效果。
TACO LLM 提供 prefix cache offload 功能，在显存 prefix cache 被驱逐时 offload 到 cpu 内存上，命中时 load 回 gpu 中，进而加速之前 prefix cache 被驱逐而得不到加速的请求。

Offload 选项

```
  --enable-prefix-cache-offload
                        Enables prefix cache offloading
  --cpu-prefill-memory-utilization CPU_PREFILL_MEMORY_UTILIZATION
                        the memory is used for prefill cache, which can
range
                        from 0 to 1. If unspecified, will use the
default
                        value of 0.3.
  --apc-offload-not-lazy
                        If set, lazy launch of layer 2~n-1 will be
disabled.
  --apc-offload-min-access-threshold APC_OFFLOAD_MIN_ACCESS_THRESHOLD
                        Min threshold for evict offloading. Default 1.
  --apc-offload-enable-hit-cnt
                        Enable hit count in APC.
  --apc-offload-gpu-evictor-limit APC_OFFLOAD_GPU_EVICTOR_LIMIT
                        The free table size limited in gpu evictor. -1
default
```

- --enable-prefix-cache-offload
  - 在开启了 APC 的基础上打开该开关，即可启用 prefix cache offload 功能。
- --cpu-prefill-memory-utilization
  - 用于 kv cache 的 cpu 内存比例，默认0.3，按卡均分。
  - Note：该比例按机器资源内存（psutil.virtual_memory.total）进行计算，与 lookahead cache 的 --cpu-decoding-memory-utilization（默认0.15）类似。请预留足够内存或配置相应比例值。
- --apc-offload-not-lazy
  - 是否关闭 offload 按层延迟启动，仅作调试用途。
- --apc-offload-min-access-threshold
  - 一个 block 会被 offload 的最小使用阈值，默认为1，即所有 block 都会被 offload。增加此值，被多次使用的 block 才会被 offload。

- --apc-offload-enable-hit-cnt
  - 打开 prefix cache offload 的命中率 log，每100个 block 打印一次。
- --apc-offload-gpu-evictor-limit
  - gpu evictor 的 free table 大小，默认为-1不生效，设置具体值限制 gpu上prefix cache 容量，仅作调试用途。

场景

- Auto Prefix Caching 适用于 common prefix 较多，prefill 计算占比较大的场景，例如 多轮对话，system prompt，代码补全等等。此类场景打开 --enable-prefix-caching 即可加速命中 gpu prefix cache 的请求的 prefill 阶段。
- GPU 容纳 prefix cache 的空间是有限的，当 common prefix 的累积量比较多，相同 common prefix 的请求间相隔比较远的场景下，之前保留的 prefix cache已经被驱逐而无法获得收益。
  - 比如 GPU 的 block 数为100，有11个不同 prefix 的请求[Q1, ..., Q11]，每个对应 prefix block 数为10，则Q11结束后Q1的 prefix cache 会被驱逐。即使Q12跟Q1的 prefix 一致，也无法获得加速。
  - 此时 prefix cache offload 通过额外的内存空间保留被驱逐的 prefix cache，进而加速Q12。
  - 可开启 --apc-offload-enable-hit-cnt 参数，通过 log cpu 判断是否有 offload 的 prefix cache 得到命中。该 log 统计所有 allocate的block 的 hit 情况。

```
[HIT CNT] total: 177800, gpu: 21944 (12.34%) cpu: 66166 (37.21%) not
hit: 89690 (50.44%)
```

量化

最近更新时间：2024-12-27 21:30:43

在本文档中，我们将介绍模型量化的基本概念，以及使用 TACO-LLM 部署量化模型的完整流程实践。

量化概述

模型量化通常是指将一个连续取值(通常是 fp32，fp16)或者大量离散值的浮点型权重，转化为有限个离散值(通常是 int8，int4)的过程。这个过程会带来轻微的推理损失精度，但是存在如下优势：

- 减小模型体积
- 降低内存占用
- 在支持低精度运算的设备上提升推理速度

1. 量化比特

工业界目前常用的量化比特位数是 4 bits 和 8 bits，低于 4bits 的量化位宽被称为低比特量化。

2. 量化目标

- **权重**：权重的量化是最常规的，量化权重可以减少模型大小和占用空间。
- **激活**：量化激活可以大大减少内存占用，结合权重的量化可以充分利用设备的算力。
- KV cache： 显存占用会随着生成的序列长度线性增长，量化 KV cache 可以节省显存，从而能够处理更大批次的大小。

量化还可以选择不同的量化粒度，例如 per-tensor，per-group 等等。并且对于激活还有动态量化和静态量化的区别。

3. 量化形式

- **线性量化**：将浮点数值域均匀的映射到整数值域，用固定的步长进行量化。该方式实现简单，硬件比较友好，适合分布相对均匀的数据。
- **非线性量化**：根据数据的实际分布特征进行不均匀的量化，在数据密集区域使用更细的量化粒度。实现和计算都比较复杂，理论上可以获得更好的量化精度。

在实际的推理业务中，由于非线性量化的计算复杂度较高，通常使用线性量化的方式。

4. 量化方法

- **量化感知训练（Quantization Aware Training， QAT）**：在训练过程中模拟量化效果，通过反向传播来补偿量化误差，让模型适应量化带来的损失。
- **训练后量化（Post Training Quantization， PTQ）**：在模型训练完成后，使用少量校准数据来确定量化参数，直接将模型量化，无需重新训练。

在实际推理业务中，PTQ 的应用更加广泛。PTQ 的主要优势在于简单和高效，但可能会引入一定程度的精度损失。

TACO-LLM 量化支持

下面展示了 TACO-LLM 在各种硬件上对不同量化方案的支持情况：

- **GPTQ**：在 Volta, Turing, Ampere, Ada, Hopper, Intel CPU 上支持。
- **AWQ**：在 Turing, Ampere, Ada, Hopper, Intel CPU 上支持。
- **Marlin**：在 Ampere, Ada, Hopper 上支持。
- **FP8**：在 Ada, Hopper 上支持。
- **Bitsandbytes**：在 Turing, Ampere, Ada, Hopper 上支持。
- **INT8(W8A8)**：在 Turing, Ampere, Ada, Hopper 上支持。
- **AQLM**：在 Volta, Turing, Ampere, Ada, Hopper 上支持。

TACO-LLM 快速启动

执行 `taco_llm serve -h` 命令可以查看 taco-llm 完整的在线模式参数配置，其中找到 quantization 的配置参数如下：

```
--quantization
{aqlm,awq,deepspeedfp,tpu_int8,fp8,fbgemm_fp8,modelopt,marlin,gguf,gptq_
marlin_24,gptq_marlin,awq_marlin,gptq,compressed-
tensors,bitsandbytes,experts_int8,qqq,neuron_quant,None}, -q
{aqlm,awq,deepspeedfp,tpu_int8,fp8,fbgemm_fp8,modelopt,marlin,gguf,gptq_
marlin_24,gptq_marlin,awq_marlin,gptq,compressed-
tensors,bitsandbytes,experts_int8,qqq,neuron_quant,None}
                        Method used to quantize the weights. If None, we
first check the `quantization_config` attribute in the model config
file. If that is None, we assume the model weights are not quantized and
use `dtype` to determine the
                        data type of the weights.
```

1. GPTQ-Marlin (AWQ)：

首先使用 AutoGPTQ(AutoAWQ) 将 fp16 模型权重量化。启动 TACO-LLM 的时候无需传入其他参数，server 会自动读取 config 文件中的量化参数来加载模型。TACO-LLM 在条件允许的情况下会默认使用 marlin kernel，可以传入 --quantization gptq 参数来强制使用 gptq kernel。

2. Bitsandbytes：

启动时添加启动参数 --quantization bitsandbytes，server 会自动读取 config 文件中的量化参数来加载模型。

3. FP8（W8A8）：

TACO-LLM 采用动态量化的方案来将 BF16/FP16 量化到 FP8，并且不需要额外的矫正数据集。除了 lm_head 的所有 linear modules 都会按照 per-tensor 的方式进行量化。

```
from taco_llm import LLM
model = LLM(moth_path, quantization="fp8")
result = model.generate("Tell me about computer science.")
```

GPTQ 量化实践（W4A16）：

本节以 TinyLlama-1.1B-Chat-v1.0量化流程为例，介绍整个量化过程。

模型量化流程

1. 首先安装 autogptq，来作为量化工具。然后下载对应的模型权重 TinyLlama-1.1B 。

```
pip install autogptq datasets transformers
```

2. 接下来可以使用下面脚本，来执行整个量化过程（其中所需的矫正数据集会自动下载）。矫正数据集可以优先使用模型对应的垂类数据集，如果没有的话，可以使用模型的预训练数据集或者是微调数据集。

```
import torch
from datasets import load_dataset
from gptqmodel import GPTQModel, QuantizeConfig
from transformers import AutoTokenizer

pretrained_model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
quantized_model_id = "TinyLlama-1.1B-Chat-v1.0-4bit-128g"

# os.makedirs(quantized_model_dir, exist_ok=True)
def get_wikitext2(tokenizer, nsamples, seqlen):
    traindata = load_dataset("wikitext", "wikitext-2-raw-v1",
split="train").filter(
        lambda x: len(x["text"]) >= seqlen)

    return [tokenizer(example["text"]) for example in
traindata.select(range(nsamples))]


@torch.no_grad()
def calculate_avg_ppl(model, tokenizer):
```

```
    from gptqmodel.utils import Perplexity

    ppl = Perplexity(
        model=model,
        tokenizer=tokenizer,
        dataset_path="wikitext",
        dataset_name="wikitext-2-raw-v1",
        split="train",
        text_column="text",
    )

    all = ppl.calculate(n_ctx=512, n_batch=512)

    # average ppl
    avg = sum(all) / len(all)

    return avg

def main():
    tokenizer = AutoTokenizer.from_pretrained(pretrained_model_id,
use_fast=True)

    traindataset = get_wikitext2(tokenizer, nsamples=256, seqlen=1024)

    quantize_config = QuantizeConfig(
        bits=4,  # quantize model to 4-bit
        group_size=128,  # it is recommended to set the value to 128
        desc_act= False, #
    )

    # load un-quantized model, the model will always be force loaded
into cpu
    model = GPTQModel.from_pretrained(pretrained_model_id,
quantize_config)

    # quantize model, the calibration_dataset should be list of dict
whose keys can only be "input_ids" and "attention_mask"
    # with value under torch.LongTensor type.
    model.quantize(traindataset)

    # save quantized model
    model.save_quantized(quantized_model_id)
```

```
    # save quantized model using safetensors
    model.save_quantized(quantized_model_id, use_safetensors=True)

    # load quantized model, currently only support cpu or single gpu
    model = GPTQModel.from_quantized(quantized_model_id,
device="cuda:0")

    # inference with model.generate
    print(tokenizer.decode(model.generate(**tokenizer("test is",
return_tensors="pt").to("cuda:0"))[0]))

    print(f"Quantized Model {quantized_model_id} avg PPL is
{calculate_avg_ppl(model, tokenizer)}")


if __name__ == "__main__":
    import logging

    logging.basicConfig(
        format="%(asctime)s %(levelname)s [%(name)s] %(message)s",
        level=logging.INFO,
        datefmt="%Y-%m-%d %H:%M:%S",
    )

    main()
```

下面介绍一下量化参数的选择：

- **--bits**： 权重量化的位宽。根据需求选择，要求节省显存选择 4，但是会对精度有较大的影响。基于显存和精度的平衡，建议选择 8，此时精度基本没有损失。
- **--group_size**：group 量化的 size 大小，越小精度越高，但是会增加推理成本。建议选择 128。
- **--desc_act**：是否使用激活重排。打开会提高量化精度，但是会增加推理成本。建议选择False。
- **--nsamples**： 矫正数据集的样本数量。数量太多会增加量化时间，且还会改变权重分布。建议选择 256。
- **--seqlen**：矫正数据集的样本长度。数量太多会增加量化时间，且还会改变权重分布。7B 模型建议选择 2048，70B 以上模型建议选择 4096。

```
bits = [4,8]
group_size = [64,128]
nsamples = [256,512]
seqlen = [2048, 4096]
```

```
desc_act = [True, False]
```

CPU 辅助加速

最近更新时间：2025-04-17 15:08:12

介绍

大模型的自回归解码的特性导致其不能充分利用 GPU 的并行计算进行加速。基于此，学术界提出了投机采样，其可以利用更高效的 Draft Model 快速生成多个 Token，然后一次性地交给 Target Model 进行验证。在接受率高的情况下，可以显著减少推理时间。
传统的投机采样使用 GPU 作为 Draft Model 的计算资源，而 GPU 的成本高昂。针对这个痛点，TACO 与 Intel 团队合作，基于 AMX 指令集对 CPU 上的矩阵乘法做了优化，借助 Intel 推出的 IPEX 加速库及其他技术对 CPU 上的推理进行加速，使得 CPU 作为 Draft Model 成为可能，从而在进行推理加速的同时，显著降低了推理成本。

> ⚠ **注意：**
> 当前该功能支持的机型：搭载 Intel EMR 8576C 或 Intel SPR 8476C 的 GPU 云服务器，其中：
> - SPR：搭载内置加速器的第四代英特尔® 至强® 可扩展处理器。
> - EMR：搭载内置加速器的第五代英特尔® 至强® 可扩展处理器。

基于 TACO-LLM 使用 CPU 单独进行推理

> ⚠ **注意：**
> 该 feature 会导致 GPU 推理的功能失效，因此若想使用 GPU 推理，需要重新安装对应的包。

1. 额外依赖安装

```
sudo apt-get update
sudo apt-get install -y libdnnl-dev
pip install intel-extension-for-pytorch==2.4.0
pip install torch --index-url https://download.pytorch.org/whl/cpu
```

2. 环境检查（如果出现问题）

需要确保：
1. torch 和 ipex 的版本号匹配
2. torch 是 cpu 版本

```
root@c2d5f7048105:/script# pip show torch
Name: torch
Version: 2.4.0+cpu
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3
Location: /usr/local/lib/python3.10/dist-packages
Requires: filelock, fsspec, jinja2, networkx, sympy, typing-extensions
Required-by: compressed-tensors, flash-attn, lightning-thunder, tensorizer, torch-tensorrt, torchaudio, torchdata, torchtext, torchvision, vllm, vllm-fla
sh-attn, xformers
root@c2d5f7048105:/script# pip show intel-extension-for-pytorch
Name: intel_extension_for_pytorch
Version: 2.4.0
Summary: Intel® Extension for PyTorch*
Home-page: https://github.com/intel/intel-extension-for-pytorch
Author: Intel Corp.
Author-email:
License: https://www.apache.org/licenses/LICENSE-2.0
Location: /usr/local/lib/python3.10/dist-packages
Requires: numpy, packaging, psutil
Required-by:
root@c2d5f7048105:/script#
```

3. 使用

以离线模式为例，其使用方法和在 GPU 上推理完全一致。

```
llm = LLM(model="facebook/opt-125m")
```

基于 TACO-LLM 使用 CPU 辅助投机采样

1. 额外依赖安装

```
sudo apt-get update
sudo apt-get install -y libdnnl-dev
pip install intel-extension-for-pytorch==2.4.0
pip install torch==2.4.0
```

2. 环境检查（如果出现问题）

需要确保：

1. torch 和 ipex 的版本号匹配
2. torch 是**不带 cpu** 的版本

```
root@b24522df0352:~# pip show torch
Name: torch
Version: 2.4.0
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3
Location: /usr/local/lib/python3.10/dist-packages
Requires: filelock, fsspec, jinja2, networkx, nvidia-cublas-cu12, nvidia-cuda-cupti-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-runt
ime-cu12, nvidia-cudnn-cu12, nvidia-cufft-cu12, nvidia-curand-cu12, nvidia-cusolver-cu12, nvidia-cusparse-cu12, nvidia-nccl-cu12,
nvidia-nvtx-cu12, sympy, triton, typing-extensions
Required-by: accelerate, compressed-tensors, flash-attn, lightning-thunder, peft, sentence-transformers, taco-llm, tensorizer, tim
m, torch-tensorrt, torchdata, torchtext, torchvision, vllm-flash-attn, xformers
root@b24522df0352:~# pip show intel-extension-for-pytorch
Name: intel_extension_for_pytorch
Version: 2.4.0
Summary: Intel® Extension for PyTorch*
Home-page: https://github.com/intel/intel-extension-for-pytorch
Author: Intel Corp.
Author-email:
License: https://www.apache.org/licenses/LICENSE-2.0
Location: /usr/local/lib/python3.10/dist-packages
Requires: numpy, packaging, psutil
Required-by:
```

3. 使用

以离线模式为例，其使用方法在投机采样原始配置上额外新增一个 cpu_draft_worker 即可。

```
llm = LLM(
      model = "meta-llama/Llama-2-7b-chat-hf",
      speculative_model = "Felladrin/Llama-68M-Chat-v1",
      num_speculative_tokens = 2,
      use_v2_block_manager = True,
      cpu_draft_worker = True,                # <<== 新增参数
 )
```

附录

AMX 介绍

什么是 AMX?

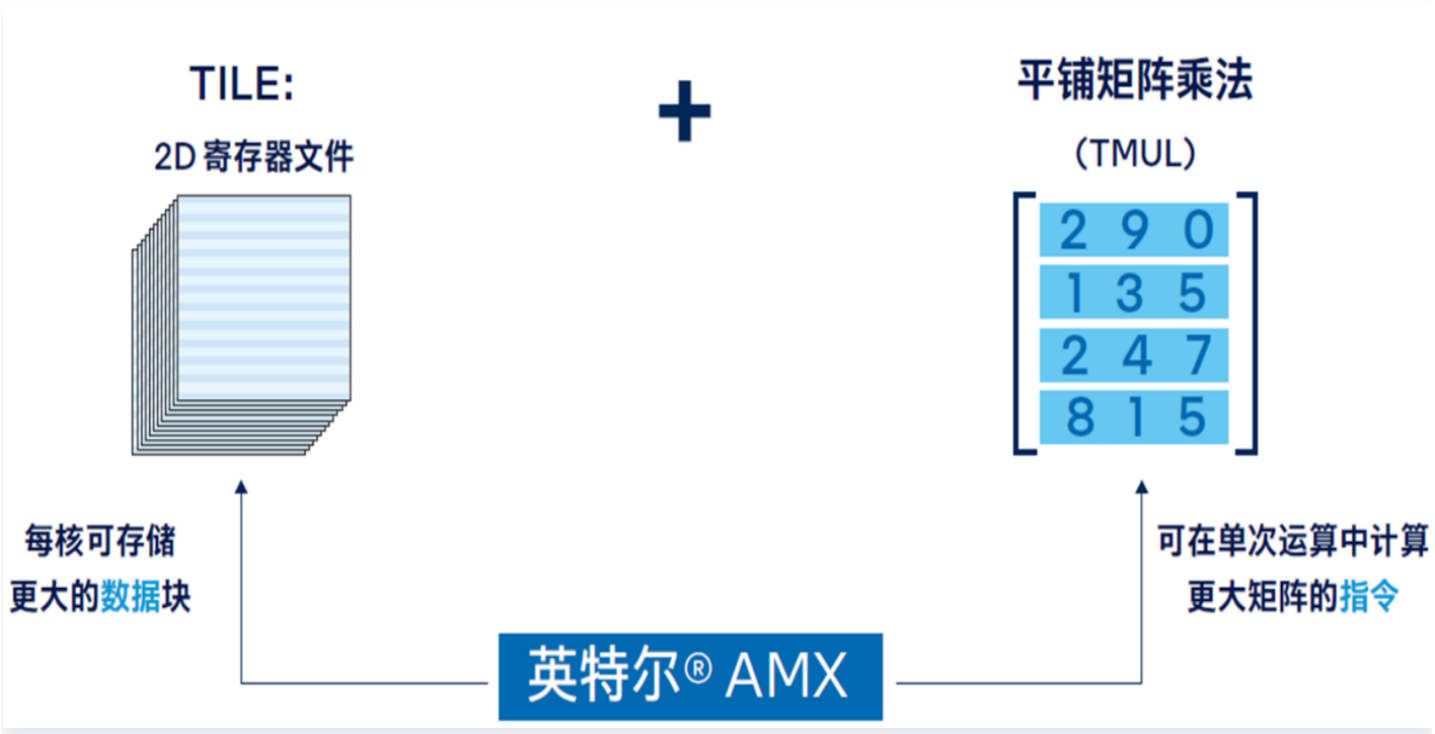
英特尔推出的第四代英特尔® 至强® 可扩展处理器及其内置的英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，英特尔® AMX）可进一步提高 AI 功能，实现较上一代产品 3 至 10 倍的推理和训练性能提升。开发人员可以编写非 AI 功能代码来利用处理器的指令集架构 (ISA)，也可编写 AI 功能代码，以充分发挥英特尔® AMX 指令集的优势。英特尔已将其 oneAPI DL 引擎——英特尔® oneAPI 深度神经网络库（Intel® oneAPI

Deep Neural Network Library，英特尔® oneDNN）集成至包括 TensorFlow、PyTorch、PaddlePaddle 和 ONNX 在内的多个主流 AI 应用开源工具当中。

AMX 架构

英特尔® AMX 架构由两部分组件构成：

- 第一部分为 TILE，由 8 个 1 KB 大小的 2D 寄存器组成，可存储大数据块。
- 第二部分为平铺矩阵乘法 (TMUL)，它是与 TILE 连接的加速引擎，可执行用于 AI 的矩阵乘法计算。



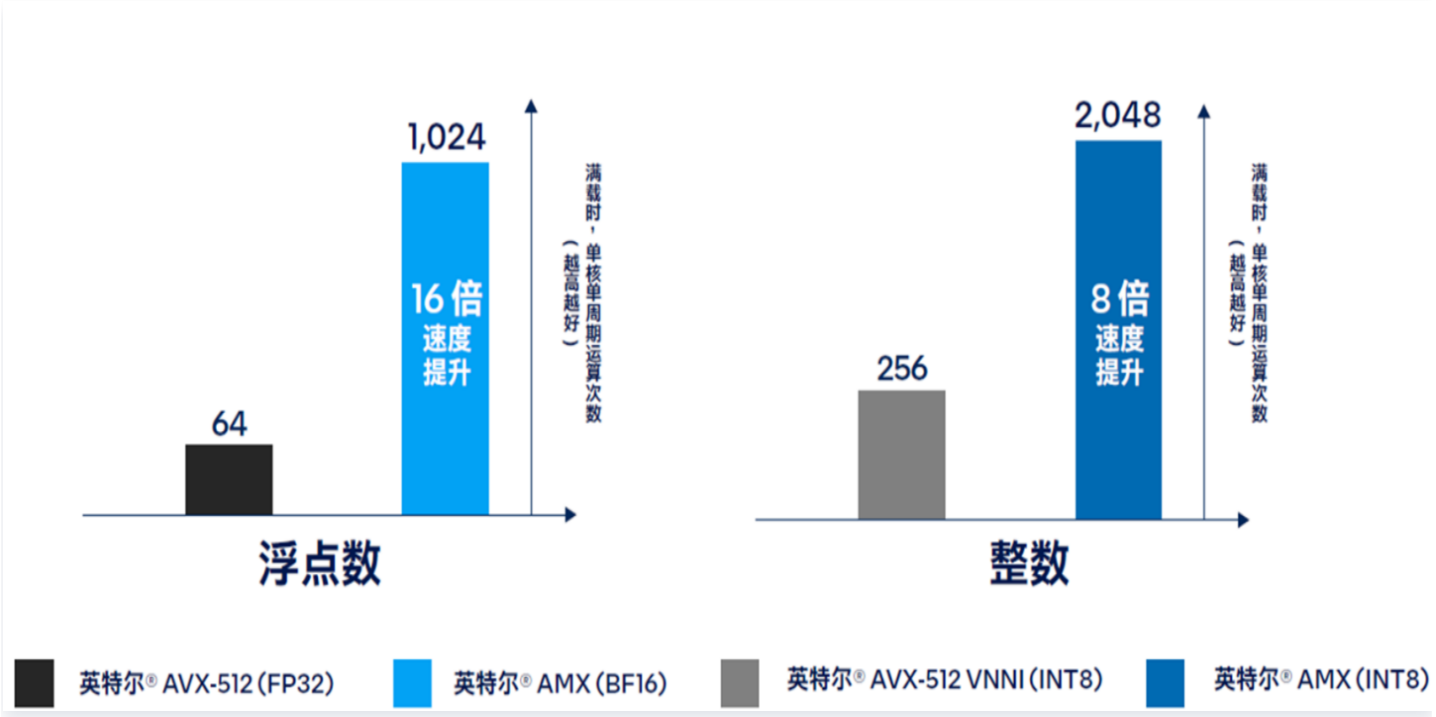
AMX 支持的数据类型

英特尔® AMX 支持两种数据类型：INT8 和 BF16，两者均可用于 AI 工作负载所需的矩阵乘法运算。

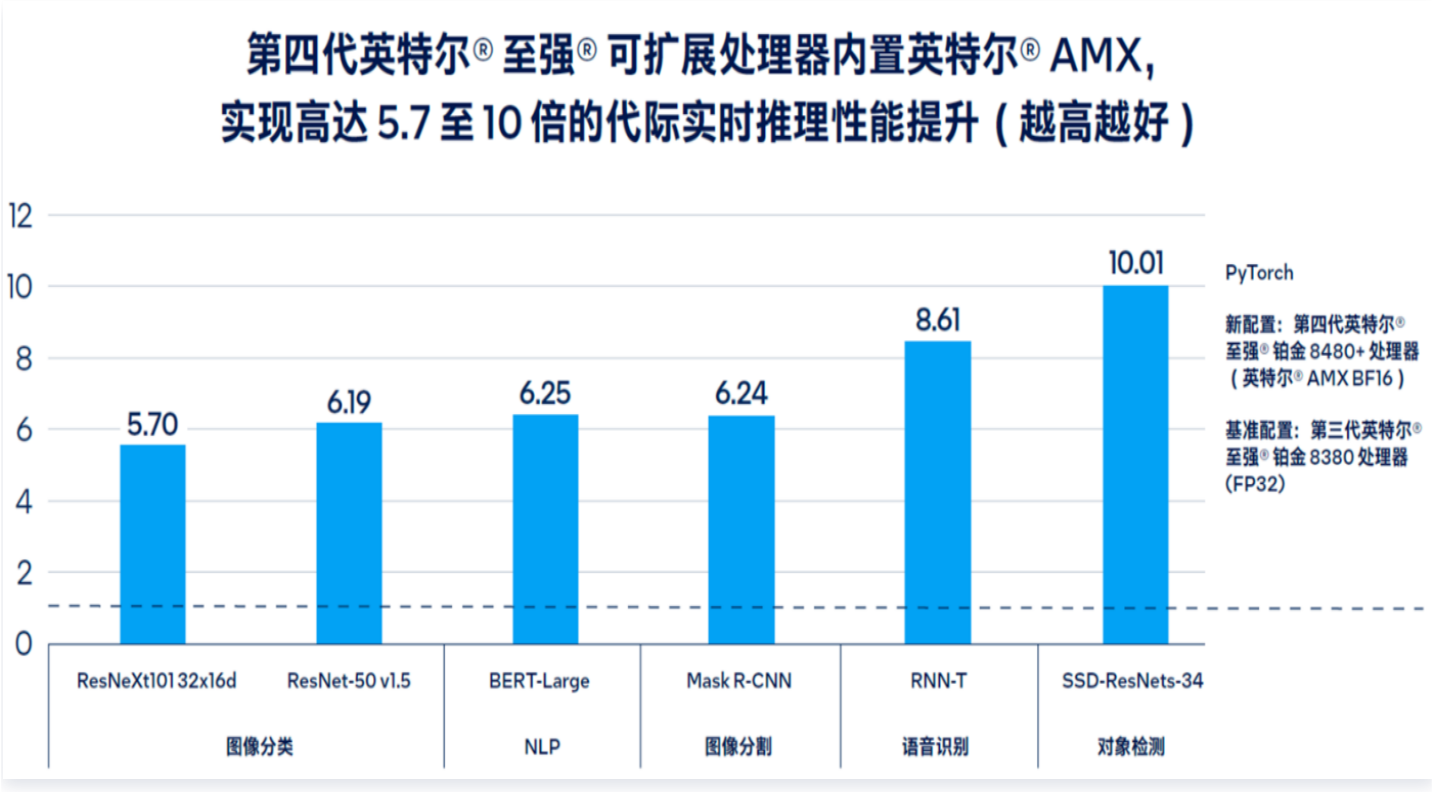
- 当推理无需 FP32 的精度时可使用 INT8 这种数据类型。由于该数据类型的精度较低，因此单位计算周期内运算次数就更多。
- BF16 这种数据类型实现的准确度足以达到大多数训练的要求，必要时它也能让 AI 推理实现更高的准确度。

AMX 的性能

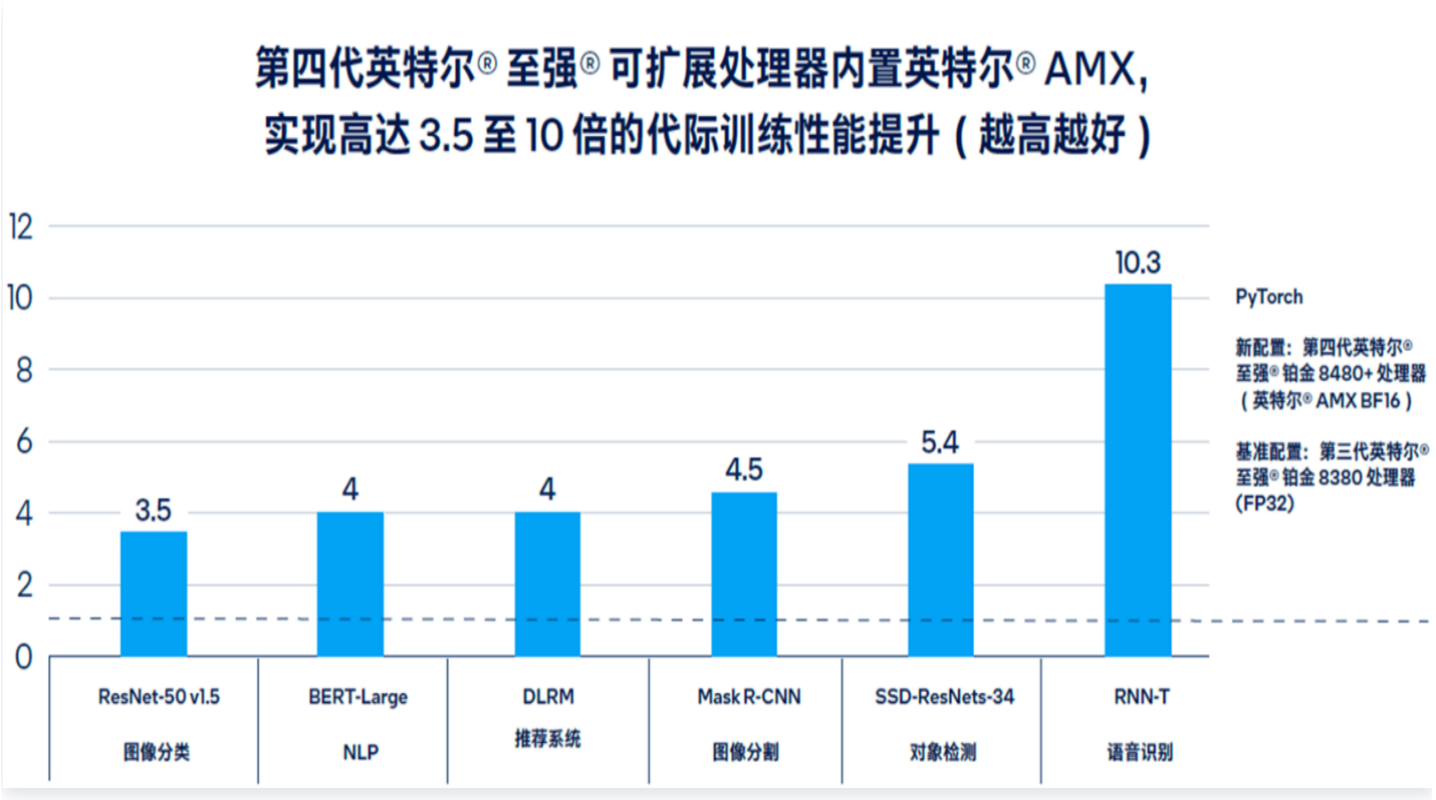
凭借这种新的平铺架构，英特尔® AMX 实现了大幅代际性能提升。与运行英特尔® 高级矢量扩展 512 神经网络指令（Intel® Advanced Vector Extensions 512 Neural Network Instructions，英特尔® AVX-512 VNNI）的第三代英特尔® 至强® 可扩展处理器相比，运行英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器将单位计算周期内执行 INT8 运算的次数从 256 次提高至 2048 次。此外，如图 6 所示，第四代英特尔® 至强® 可扩展处理器可在单位计算周期内执行 1024 次 BF16 运算，而第三代英特尔® 至强® 可扩展处理器执行 FP32 运算的次数仅为 64 次。



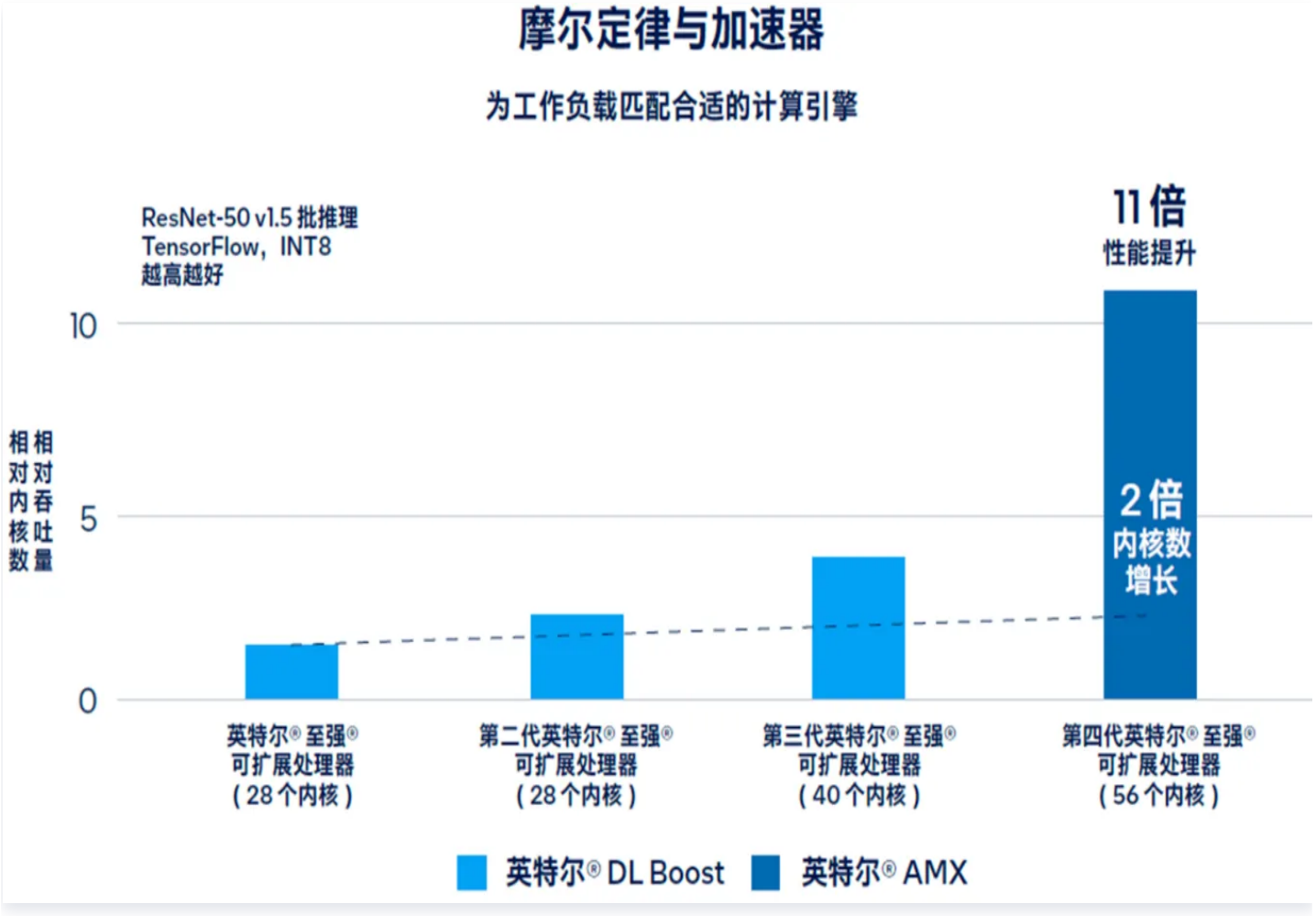
下图所示为英特尔® AMX 在代际间实现高达 5.7 至 10 倍的 PyTorch 实时推理性能提升的情况。



下图所示为英特尔® AMX 在代际间实现高达 3.5 至 10 倍的 PyTorch 训练性能提升的情况。



通过下图可以看出英特尔® AMX 带来的性能提升远大于每一代产品（从第一代英特尔® 至强® 可扩展处理器开始）通过增加内核所实现的性能提升。



IPEX 介绍

Intel® Extension for PyTorch* (IPEX) 是由英特尔发起的一个开源扩展项目，旨在通过模块级的全面优化及更为简洁的API接口，在基于原生PyTorch框架运行时，显著提升深度学习任务在英特尔硬件（包括但不限于CPU和GPU）上的推理与训练性能。IPEX兼容PyTorch生态系统中超过90%的主流模型，并对其中50多个深度模型进行了特别优化。用户只需添加少量代码以启用BF16混合精度支持，即可轻松享受到性能上的显著改进，整个过程无需复杂的配置调整，从而提供了近乎即插即用的便捷体验。

此外，Intel® Extension for PyTorch* 通过对Intel硬件特性的深入挖掘，如利用Intel® Advanced Vector Extensions 512 (AVX-512) 中的向量神经网络指令(VNNI)、Intel® Advanced Matrix Extensions (AMX) 以及Intel独立显卡上配备的Intel Xe Matrix Extensions (XMX) AI加速引擎等技术，进一步增强了PyTorch在Intel平台上的执行效率。

长序列优化

最近更新时间：2025-08-04 10:53:21

背景

在 LLM 大模型推理中，长序列场景应用越来越广泛，目前业界对长序列的优化主要是 kv cache 量化、稀疏化等方法，这些都对模型精度有一定的影响。TACO LLM 自研长序列并行方案，可以在长序列场景进行精度无损的加速。长序列推理场景首字延迟会比较慢，针对该问题，序列并行在 prefill 阶段采用了 Ring Attention 类的方案，可以通过扩展机器，降低首字延迟。而对于推理阶段，TACO LLM 可以使用 Lookahead 等投机采样技术加速。

使用方式

启动 TACO LLM 服务时添加参数 **--sequence-parallel-size 2** 即可开启序列并行，目前只支持 serve 方式，不支持 LLM 的离线方式（0.6.4版本开始支持）。

最佳实践

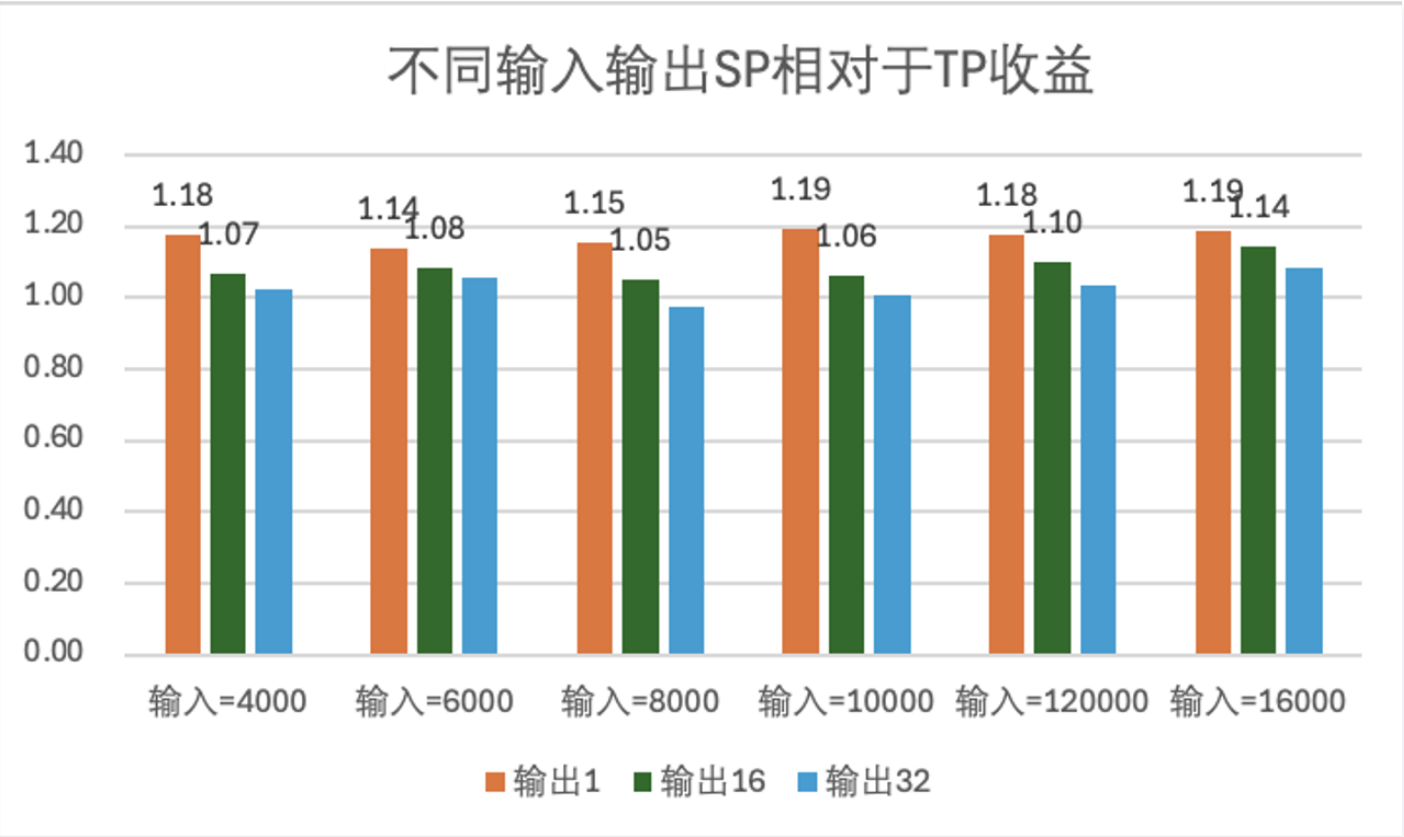
序列并行可以和张量并行一起使用，同时也可以和 FP8一起使用。序列并行适合符合如下几个特点的场景：

- 由于序列并行具备通信计算重叠的优点，故适合在 PNV5b、4090PCIe 系列卡上。
- 适合输入较长，输出较短的场景。
- 建议与 FP8 一起使用，加速效果更好。
- 适合 GQA 类的模型，由于 kv cache 更小，通信更小，相对于 TP 来说加速更快。

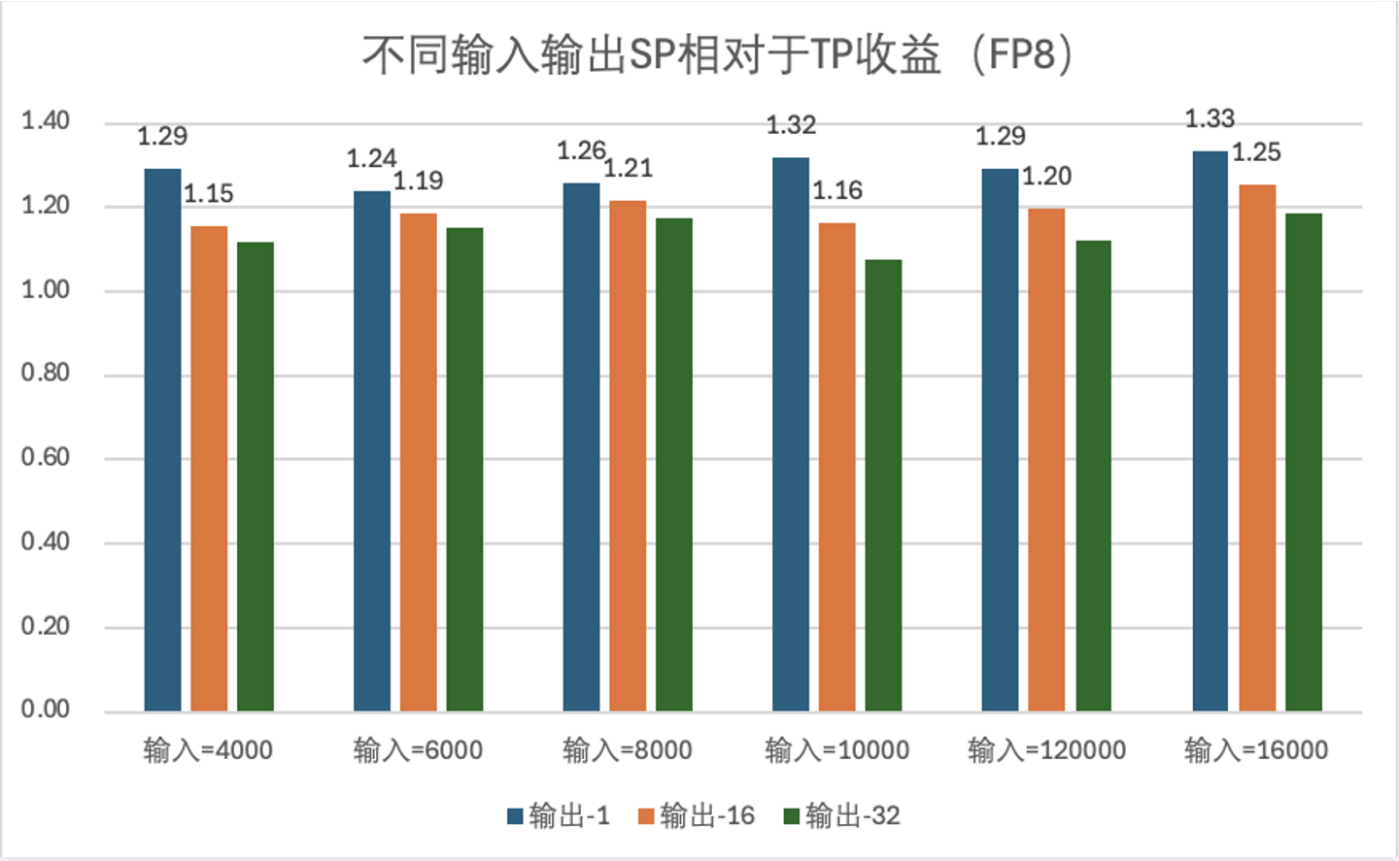
性能数据

MHA 场景

当前测试模型大小为 Llama2 7B，序列长度为6000， SP 为2，在 PNV6/HCCPNV6 和 A800上，相对于 TP 无性能提升。
在 PNV5b 上 FP16：SP = 2的场景相对于 TP = 2场景，端到端性能有5%到15%的提升，prefill 性能提升 15%-20%。



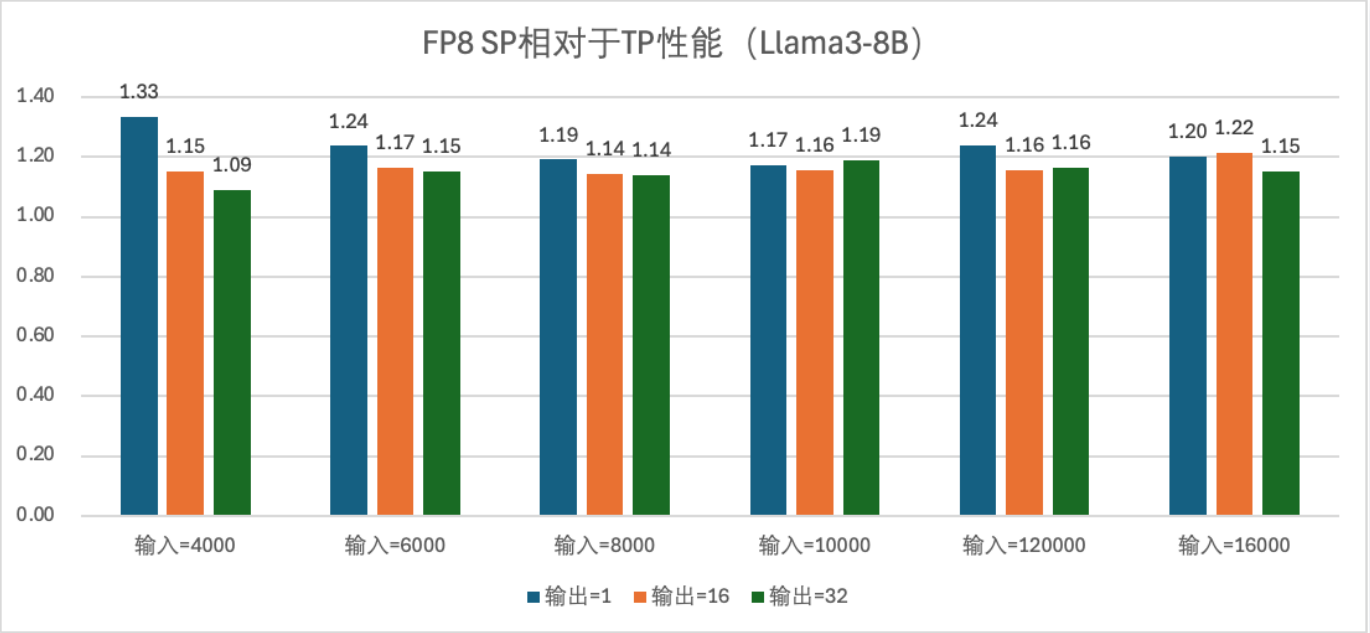
在 PNV5b 上 FP8：SP = 2的场景相对于 TP = 2场景，性能有15%到25%的提升。



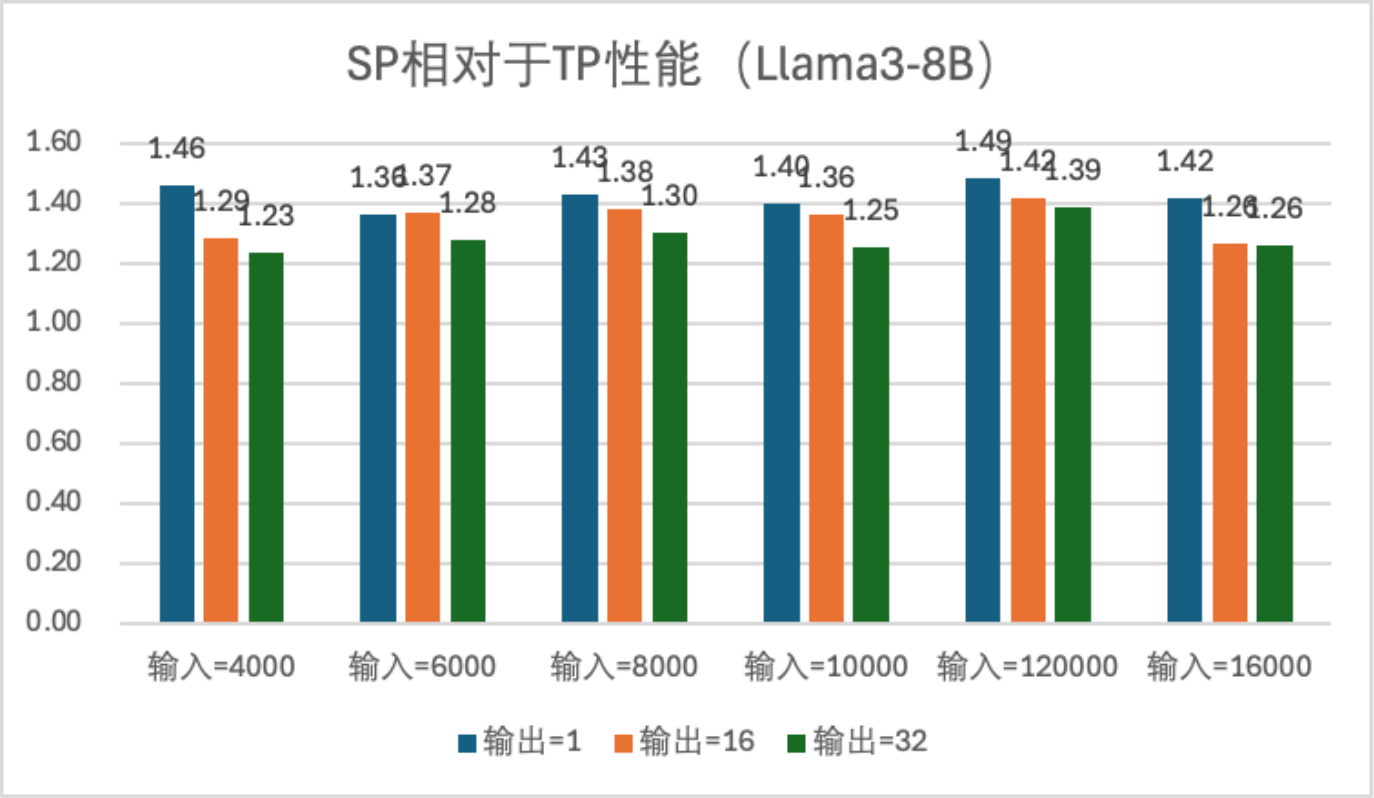
GQA 场景

当前测试模型大小为 Llama3-8B。

在 PNV5b 上 FP16：SP = 2的场景相对于 TP = 2场景，端到端性能有10%到20%的提升，prefill 性能提升20%-30%。



在 PNV5b 上 FP8：SP = 2的场景相对于 TP = 2场景，端到端性能有20%到40%的提升，prefill 性能有40%-50%。



TACO LLM API
Offline API

最近更新时间：2024-12-27 21:30:43

LLM

LLM 构造参数

```
class taco_llm.LLM(
    model: str,
    tokenizer: Optional[str] = None,
    tokenizer_mode: str = "auto",
    skip_tokenizer_init: bool = False,
    trust_remote_code: bool = False,
    tensor_parallel_size: int = 1,
    dtype: str = "auto",
    quantization: Optional[str] = None,
    revision: Optional[str] = None,
    tokenizer_revision: Optional[str] = None,
    seed: int = 0,
    gpu_memory_utilization: float = 0.9,
    swap_space: float = 4,
    cpu_offload_gb: float = 0,
    enforce_eager: Optional[bool] = None,
    max_context_len_to_capture: Optional[int] = None,
    max_seq_len_to_capture: int = 8192,
    disable_custom_all_reduce: bool = False,
    disable_async_output_proc: bool = False,
    **kwargs,
)
    """
    This class includes a tokenizer, a language model (possibly
distributed
    across multiple GPUs), and GPU memory space allocated for
intermediate
    states (aka KV cache). Given a batch of prompts and sampling
parameters,
    this class generates texts from the model, using an intelligent
batching
    mechanism and efficient memory management.
```

```
    Args:
        model: The name or path of a HuggingFace Transformers model.
        tokenizer: The name or path of a HuggingFace Transformers
tokenizer.
        tokenizer_mode: The tokenizer mode. "auto" will use the fast
tokenizer
            if available, and "slow" will always use the slow tokenizer.
        skip_tokenizer_init: If true, skip initialization of tokenizer
and
            detokenizer. Expect valid prompt_token_ids and None for
prompt
            from the input.
        trust_remote_code: Trust remote code (e.g., from HuggingFace)
when
            downloading the model and tokenizer.
        tensor_parallel_size: The number of GPUs to use for distributed
            execution with tensor parallelism.
        dtype: The data type for the model weights and activations.
Currently,
            we support float32, float16, and bfloat16. If auto, we use
            the torch_dtype attribute specified in the model config
file.
            However, if the torch_dtype in the config is float32, we
will
            use float16 instead.
        quantization: The method used to quantize the model weights.
Currently,
            we support "awq", "gptq", and "fp8" (experimental).
            If None, we first check the quantization_config attribute in
the
            model config file. If that is None, we assume the model
weights are
            not quantized and use dtype to determine the data type of
            the weights.
        revision: The specific model version to use. It can be a branch
name,
            a tag name, or a commit id.
        tokenizer_revision: The specific tokenizer version to use. It
can be a
            branch name, a tag name, or a commit id.
        seed: The seed to initialize the random number generator for
sampling.
```

```
        gpu_memory_utilization: The ratio (between 0 and 1) of GPU
memory to
            reserve for the model weights, activations, and KV cache.
Higher
            values will increase the KV cache size and thus improve the
model's
            throughput. However, if the value is too high, it may cause
out-of-
            memory (OOM) errors.
        swap_space: The size (GiB) of CPU memory per GPU to use as swap
space.
            This can be used for temporarily storing the states of the
requests
            when their best_of sampling parameters are larger than 1. If
all
            requests will have best_of=1, you can safely set this to 0.
            Otherwise, too small values may cause out-of-memory (OOM)
errors.
        cpu_offload_gb: The size (GiB) of CPU memory to use for
offloading
            the model weights. This virtually increases the GPU memory
space
            you can use to hold the model weights, at the cost of CPU-
GPU data
            transfer for every forward pass.
        enforce_eager: Whether to enforce eager execution. If True, we
will
            disable CUDA graph and always execute the model in eager
mode.
            If False, we will use CUDA graph and eager execution in
hybrid.
        max_context_len_to_capture: Maximum context len covered by CUDA
graphs.
            When a sequence has context length larger than this, we fall
back
            to eager mode (DEPRECATED. Use max_seq_len_to_capture
instead).
        max_seq_len_to_capture: Maximum sequence len covered by CUDA
graphs.
            When a sequence has context length larger than this, we fall
back
            to eager mode.
        disable_custom_all_reduce: See ParallelConfig
```

```
        **kwargs: Arguments for :class:taco_llm.EngineArgs.

    Note:
        This class is intended to be used for offline inference. For
online
        serving, use the :class:taco_llm.AsyncLLMEngine class instead.
    """
```

TACO-LLM 支持离线和在线两种模式，这两种模式的参数配置是一致的。因此，除了上述明确提到的参数外，您还可以设置任意 TACO-LLM 在线模式支持的参数。完整的参数配置请参见 在线模式 API 章节。

chat 接口

```
def chat(
    self,
    messages: List[ChatCompletionMessageParam],
    sampling_params: Optional[Union[SamplingParams,
                                    List[SamplingParams]]] = None,
    use_tqdm: bool = True,
    lora_request: Optional[LoRARequest] = None,
    chat_template: Optional[str] = None,
    add_generation_prompt: bool = True,
) -> List[RequestOutput]:
    """
    Generate responses for a chat conversation.

    The chat conversation is converted into a text prompt using the
    tokenizer and calls the :meth:`generate` method to generate the
    responses.

    Multi-modal inputs can be passed in the same way you would pass them
    to the OpenAI API.

    Args:
        messages: A single conversation represented as a list of
messages.
            Each message is a dictionary with 'role' and 'content' keys.
        sampling_params: The sampling parameters for text generation.
            If None, we use the default sampling parameters. When it
            is a single value, it is applied to every prompt. When it
            is a list, the list must have the same length as the
            prompts and it is paired one by one with the prompt.
```

```
        use_tqdm: Whether to use tqdm to display the progress bar.
        lora_request: LoRA request to use for generation, if any.
        chat_template: The template to use for structuring the chat.
          If not provided, the model's default chat template will be
used.
        add_generation_prompt: If True, adds a generation template
            to each message.

    Returns:
        A list of ``RequestOutput`` objects containing the generated
        responses in the same order as the input messages.
    """
```

generate 接口

```
def generate(
    self,
    prompts: Union[Union[PromptInputs, Sequence[PromptInputs]],
                   Optional[Union[str, List[str]]]] = None,
    sampling_params: Optional[Union[SamplingParams,
                                    Sequence[SamplingParams]]] = None,
    prompt_token_ids: Optional[Union[List[int], List[List[int]]]] =
None,
    use_tqdm: bool = True,
    lora_request: Optional[Union[List[LoRARequest], LoRARequest]] =
None,
    prompt_adapter_request: Optional[PromptAdapterRequest] = None,
    guided_options_request: Optional[Union[LLMGuidedOptions,
                                           GuidedDecodingRequest]] =
None
) -> List[RequestOutput]:
    """Generates the completions for the input prompts.
    This class automatically batches the given prompts, considering
    the memory constraint. For the best performance, put all of your
prompts
    into a single list and pass it to this method.

    Args:
        inputs: A list of inputs to generate completions for.
        sampling_params: The sampling parameters for text generation. If
            None, we use the default sampling parameters.
            When it is a single value, it is applied to every prompt.
            When it is a list, the list must have the same length as the
```

```
            prompts and it is paired one by one with the prompt.
        use_tqdm: Whether to use tqdm to display the progress bar.
        lora_request: LoRA request to use for generation, if any.
        prompt_adapter_request: Prompt Adapter request to use for
            generation, if any.

    Returns:
        A list of ``RequestOutput`` objects containing the
        generated completions in the same order as the input prompts.

    Note:
        Using ``prompts`` and ``prompt_token_ids`` as keyword parameters
is
        considered legacy and may be deprecated in the future. You
should
        instead pass them via the ``inputs`` parameter.
    """
```

Online API

最近更新时间：2024-12-27 21:30:43

执行命令 `taco_llm serve -h` 可以查看 TACO-LLM 的完整在线模式参数配置：

```
# taco_llm serve -h

usage: taco_llm serve <model_tag> [options]

positional arguments:
  model_tag             The model tag to serve

options:
  -h, --help            show this help message and exit
  --config CONFIG       Read CLI options from a config file.Must be a
YAML with the following
options:https://docs.vllm.ai/en/latest/serving/openai_compatible_server.
html#command-line-arguments-for-the-server
  --host HOST           host name
  --port PORT           port number
  --uvicorn-log-level {debug,info,warning,error,critical,trace}
                        log level for uvicorn
  --allow-credentials   allow credentials
  --allowed-origins ALLOWED_ORIGINS
                        allowed origins
  --allowed-methods ALLOWED_METHODS
                        allowed methods
  --allowed-headers ALLOWED_HEADERS
                        allowed headers
  --api-key API_KEY     If provided, the server will require this key to
be presented in the header.
  --lora-modules LORA_MODULES [LORA_MODULES ...]
                        LoRA module configurations in the format
name=path. Multiple modules can be specified.
  --prompt-adapters PROMPT_ADAPTERS [PROMPT_ADAPTERS ...]
                        Prompt adapter configurations in the format
name=path. Multiple adapters can be specified.
  --chat-template CHAT_TEMPLATE
                        The file path to the chat template, or the
template in single-line form for the specified model
  --response-role RESPONSE_ROLE
```

```
                        The role name to return if
request.add_generation_prompt=true.
  --ssl-keyfile SSL_KEYFILE
                        The file path to the SSL key file
  --ssl-certfile SSL_CERTFILE
                        The file path to the SSL cert file
  --ssl-ca-certs SSL_CA_CERTS
                        The CA certificates file
  --ssl-cert-reqs SSL_CERT_REQS
                        Whether client certificate is required (see
stdlib ssl module's)
  --root-path ROOT_PATH
                        FastAPI root_path when app is behind a path
based routing proxy
  --middleware MIDDLEWARE
                        Additional ASGI middleware to apply to the app.
We accept multiple --middleware arguments. The value should be an import
path. If a function is provided, taco_llm will add it to the server
using @app.middleware('http').
                        If a class is provided, taco_llm will add it to
the server using app.add_middleware().
  --return-tokens-as-token-ids
                        When --max-logprobs is specified, represents
single tokens as strings of the form 'token_id:{token_id}' so that
tokens that are not JSON-encodable can be identified.
  --disable-frontend-multiprocessing
                        If specified, will run the OpenAI frontend
server in the same process as the model serving engine.
  --enable-auto-tool-choice
                        Enable auto tool choice for supported models.
Use --tool-call-parserto specify which parser to use
  --tool-call-parser {mistral,hermes}
                        Select the tool call parser depending on the
model that you're using. This is used to parse the model-generated tool
call into OpenAI API format. Required for --enable-auto-tool-choice.
  --model MODEL         Name or path of the huggingface model to use.
  --tokenizer TOKENIZER
                        Name or path of the huggingface tokenizer to
use. If unspecified, model name or path will be used.
  --skip-tokenizer-init
                        Skip initialization of tokenizer and detokenizer
  --revision REVISION   The specific model version to use. It can be a
branch name, a tag name, or a commit id. If unspecified, will use the
```

```
default version.
  --code-revision CODE_REVISION
                        The specific revision to use for the model code
on Hugging Face Hub. It can be a branch name, a tag name, or a commit
id. If unspecified, will use the default version.
  --tokenizer-revision TOKENIZER_REVISION
                        Revision of the huggingface tokenizer to use. It
can be a branch name, a tag name, or a commit id. If unspecified, will
use the default version.
  --tokenizer-mode {auto,slow,mistral}
                        The tokenizer mode. * "auto" will use the fast
tokenizer if available. * "slow" will always use the slow tokenizer. *
"mistral" will always use the mistral_common tokenizer.
  --trust-remote-code   Trust remote code from huggingface.
  --download-dir DOWNLOAD_DIR
                        Directory to download and load the weights,
default to the default cache dir of huggingface.
  --load-format
{auto,pt,safetensors,npcache,dummy,tensorizer,sharded_state,gguf,bitsand
bytes,mistral}
                        The format of the model weights to load. *
"auto" will try to load the weights in the safetensors format and fall
back to the pytorch bin format if safetensors format is not available. *
"pt" will load the weights in the
                        pytorch bin format. * "safetensors" will load
the weights in the safetensors format. * "npcache" will load the weights
in pytorch format and store a numpy cache to speed up the loading. *
"dummy" will initialize the
                        weights with random values, which is mainly for
profiling. * "tensorizer" will load the weights using tensorizer from
CoreWeave. See the Tensorize vLLM Model script in the Examples section
for more information. *
                        "bitsandbytes" will load the weights using
bitsandbytes quantization.
  --config-format {auto,hf,mistral}
                        The format of the model config to load. * "auto"
will try to load the config in hf format if available else it will try
to load in mistral format
  --dtype {auto,half,float16,bfloat16,float,float32}
                        Data type for model weights and activations. *
"auto" will use FP16 precision for FP32 and FP16 models, and BF16
precision for BF16 models. * "half" for FP16. Recommended for AWQ
quantization. * "float16" is the same as
```

```
                        "half". * "bfloat16" for a balance between
precision and range. * "float" is shorthand for FP32 precision. *
"float32" for FP32 precision.
  --kv-cache-dtype {auto,fp8,fp8_e5m2,fp8_e4m3}
                        Data type for kv cache storage. If "auto", will
use model data type. CUDA 11.8+ supports fp8 (=fp8_e4m3) and fp8_e5m2.
ROCm (AMD GPU) supports fp8 (=fp8_e4m3)
  --quantization-param-path QUANTIZATION_PARAM_PATH
                        Path to the JSON file containing the KV cache
scaling factors. This should generally be supplied, when KV cache dtype
is FP8. Otherwise, KV cache scaling factors default to 1.0, which may
cause accuracy issues. FP8_E5M2
                        (without scaling) is only supported on cuda
versiongreater than 11.8. On ROCm (AMD GPU), FP8_E4M3 is instead
supported for common inference criteria.
  --max-model-len MAX_MODEL_LEN
                        Model context length. If unspecified, will be
automatically derived from the model config.
  --guided-decoding-backend {outlines,lm-format-enforcer}
                        Which engine will be used for guided decoding
(JSON schema / regex etc) by default. Currently support
https://github.com/outlines-dev/outlines and
https://github.com/noamgat/lm-format-enforcer. Can be overridden per
                        request via guided_decoding_backend parameter.
  --distributed-executor-backend {ray,mp}
                        Backend to use for distributed serving. When
more than 1 GPU is used, will be automatically set to "ray" if installed
or "mp" (multiprocessing) otherwise.
  --worker-use-ray      Deprecated, use --distributed-executor-
backend=ray.
  --pipeline-parallel-size PIPELINE_PARALLEL_SIZE, -pp
PIPELINE_PARALLEL_SIZE
                        Number of pipeline stages.
  --tensor-parallel-size TENSOR_PARALLEL_SIZE, -tp TENSOR_PARALLEL_SIZE
                        Number of tensor parallel replicas.
  --max-parallel-loading-workers MAX_PARALLEL_LOADING_WORKERS
                        Load model sequentially in multiple batches, to
avoid RAM OOM when using tensor parallel and large models.
  --ray-workers-use-nsight
                        If specified, use nsight to profile Ray workers.
  --block-size {8,16,32}
                        Token block size for contiguous chunks of
tokens. This is ignored on neuron devices and set to max-model-len
```

```
  --enable-prefix-caching
                        Enables automatic prefix caching.
  --disable-sliding-window
                        Disables sliding window, capping to sliding
window size
  --use-v2-block-manager
                        Use BlockSpaceMangerV2.
  --num-lookahead-slots NUM_LOOKAHEAD_SLOTS
                        Experimental scheduling config necessary for
speculative decoding. This will be replaced by speculative config in the
future; it is present to enable correctness tests until then.
  --seed SEED           Random seed for operations.
  --swap-space SWAP_SPACE
                        CPU swap space size (GiB) per GPU.
  --cpu-offload-gb CPU_OFFLOAD_GB
                        The space in GiB to offload to CPU, per GPU.
Default is 0, which means no offloading. Intuitively, this argument can
be seen as a virtual way to increase the GPU memory size. For example,
if you have one 24 GB GPU and set
                        this to 10, virtually you can think of it as a
34 GB GPU. Then you can load a 13B model with BF16 weight,which requires
at least 26GB GPU memory. Note that this requires fast CPU-GPU
interconnect, as part of the model
                        isloaded from CPU memory to GPU memory on the
fly in each model forward pass.
  --gpu-memory-utilization GPU_MEMORY_UTILIZATION
                        The fraction of GPU memory to be used for the
model executor, which can range from 0 to 1. For example, a value of 0.5
would imply 50% GPU memory utilization. If unspecified, will use the
default value of 0.9.
  --num-gpu-blocks-override NUM_GPU_BLOCKS_OVERRIDE
                        If specified, ignore GPU profiling result and
use this numberof GPU blocks. Used for testing preemption.
  --max-num-batched-tokens MAX_NUM_BATCHED_TOKENS
                        Maximum number of batched tokens per iteration.
  --max-num-seqs MAX_NUM_SEQS
                        Maximum number of sequences per iteration.
  --max-logprobs MAX_LOGPROBS
                        Max number of log probs to return logprobs is
specified in SamplingParams.
  --disable-log-stats   Disable logging statistics.
  --quantization
{aqlm,awq,deepspeedfp,tpu_int8,fp8,fbgemm_fp8,modelopt,marlin,gguf,gptq_
```

```
marlin_24,gptq_marlin,awq_marlin,gptq,compressed-
tensors,bitsandbytes,experts_int8,qqq,neuron_quant,None}, -q
{aqlm,awq,deepspeedfp,tpu_int8,fp8,fbgemm_fp8,modelopt,marlin,gguf,gptq_
marlin_24,gptq_marlin,awq_marlin,gptq,compressed-
tensors,bitsandbytes,experts_int8,qqq,neuron_quant,None}
                        Method used to quantize the weights. If None, we
first check the quantization_config attribute in the model config file.
If that is None, we assume the model weights are not quantized and use
dtype to determine the
                        data type of the weights.
  --rope-scaling ROPE_SCALING
                        RoPE scaling configuration in JSON format. For
example, {"type":"dynamic","factor":2.0}
  --rope-theta ROPE_THETA
                        RoPE theta. Use with rope_scaling. In some
cases, changing the RoPE theta improves the performance of the scaled
model.
  --enforce-eager       Always use eager-mode PyTorch. If False, will
use eager mode and CUDA graph in hybrid for maximal performance and
flexibility.
  --max-context-len-to-capture MAX_CONTEXT_LEN_TO_CAPTURE
                        Maximum context length covered by CUDA graphs.
When a sequence has context length larger than this, we fall back to
eager mode. (DEPRECATED. Use --max-seq-len-to-capture instead)
  --max-seq-len-to-capture MAX_SEQ_LEN_TO_CAPTURE
                        Maximum sequence length covered by CUDA graphs.
When a sequence has context length larger than this, we fall back to
eager mode.
  --disable-custom-all-reduce
                        See ParallelConfig.
  --tokenizer-pool-size TOKENIZER_POOL_SIZE
                        Size of tokenizer pool to use for asynchronous
tokenization. If 0, will use synchronous tokenization.
  --tokenizer-pool-type TOKENIZER_POOL_TYPE
                        Type of tokenizer pool to use for asynchronous
tokenization. Ignored if tokenizer_pool_size is 0.
  --tokenizer-pool-extra-config TOKENIZER_POOL_EXTRA_CONFIG
                        Extra config for tokenizer pool. This should be
a JSON string that will be parsed into a dictionary. Ignored if
tokenizer_pool_size is 0.
  --limit-mm-per-prompt LIMIT_MM_PER_PROMPT
                        For each multimodal plugin, limit how many input
instances to allow for each prompt. Expects a comma-separated list of
```

```
items, e.g.: image=16,video=2 allows a maximum of 16 images and 2 videos
per prompt. Defaults to 1
                        for each modality.
  --enable-lora         If True, enable handling of LoRA adapters.
  --max-loras MAX_LORAS
                        Max number of LoRAs in a single batch.
  --max-lora-rank MAX_LORA_RANK
                        Max LoRA rank.
  --lora-extra-vocab-size LORA_EXTRA_VOCAB_SIZE
                        Maximum size of extra vocabulary that can be
present in a LoRA adapter (added to the base model vocabulary).
  --lora-dtype {auto,float16,bfloat16,float32}
                        Data type for LoRA. If auto, will default to
base model dtype.
  --long-lora-scaling-factors LONG_LORA_SCALING_FACTORS
                        Specify multiple scaling factors (which can be
different from base model scaling factor - see eg. Long LoRA) to allow
for multiple LoRA adapters trained with those scaling factors to be used
at the same time. If not
                        specified, only adapters trained with the base
model scaling factor are allowed.
  --max-cpu-loras MAX_CPU_LORAS
                        Maximum number of LoRAs to store in CPU memory.
Must be >= than max_num_seqs. Defaults to max_num_seqs.
  --fully-sharded-loras
                        By default, only half of the LoRA computation is
sharded with tensor parallelism. Enabling this will use the fully
sharded layers. At high sequence length, max rank or tensor parallel
size, this is likely faster.
  --enable-prompt-adapter
                        If True, enable handling of PromptAdapters.
  --max-prompt-adapters MAX_PROMPT_ADAPTERS
                        Max number of PromptAdapters in a batch.
  --max-prompt-adapter-token MAX_PROMPT_ADAPTER_TOKEN
                        Max number of PromptAdapters tokens
  --device {auto,cuda,neuron,cpu,openvino,tpu,xpu}
                        Device type for vLLM execution.
  --num-scheduler-steps NUM_SCHEDULER_STEPS
                        Maximum number of forward steps per scheduler
call.
  --scheduler-delay-factor SCHEDULER_DELAY_FACTOR
                        Apply a delay (of delay factor multiplied by
previousprompt latency) before scheduling next prompt.
```

```
  --enable-chunked-prefill [ENABLE_CHUNKED_PREFILL]
                        If set, the prefill requests can be chunked
based on the max_num_batched_tokens.
  --speculative-model SPECULATIVE_MODEL
                        The name of the draft model to be used in
speculative decoding.
  --speculative-model-quantization
{aqlm,awq,deepspeedfp,tpu_int8,fp8,fbgemm_fp8,modelopt,marlin,gguf,gptq_
marlin_24,gptq_marlin,awq_marlin,gptq,compressed-
tensors,bitsandbytes,experts_int8,qqq,neuron_quant,None}
                        Method used to quantize the weights of
speculative model.If None, we first check the quantization_config
attribute in the model config file. If that is None, we assume the model
weights are not quantized and use dtype
                        to determine the data type of the weights.
  --num-speculative-tokens NUM_SPECULATIVE_TOKENS
                        The number of speculative tokens to sample from
the draft model in speculative decoding.
  --speculative-draft-tensor-parallel-size
SPECULATIVE_DRAFT_TENSOR_PARALLEL_SIZE, -spec-draft-tp
SPECULATIVE_DRAFT_TENSOR_PARALLEL_SIZE
                        Number of tensor parallel replicas for the draft
model in speculative decoding.
  --speculative-max-model-len SPECULATIVE_MAX_MODEL_LEN
                        The maximum sequence length supported by the
draft model. Sequences over this length will skip speculation.
  --speculative-disable-by-batch-size SPECULATIVE_DISABLE_BY_BATCH_SIZE
                        Disable speculative decoding for new incoming
requests if the number of enqueue requests is larger than this value.
  --ngram-prompt-lookup-max NGRAM_PROMPT_LOOKUP_MAX
                        Max size of window for ngram prompt lookup in
speculative decoding.
  --ngram-prompt-lookup-min NGRAM_PROMPT_LOOKUP_MIN
                        Min size of window for ngram prompt lookup in
speculative decoding.
  --spec-decoding-acceptance-method
{rejection_sampler,typical_acceptance_sampler}
                        Specify the acceptance method to use during
draft token verification in speculative decoding. Two types of
acceptance routines are supported: 1) RejectionSampler which does not
allow changing the acceptance rate of draft
                        tokens, 2) TypicalAcceptanceSampler which is
configurable, allowing for a higher acceptance rate at the cost of lower
```

```
quality, and vice versa.
  --typical-acceptance-sampler-posterior-threshold
TYPICAL_ACCEPTANCE_SAMPLER_POSTERIOR_THRESHOLD
                        Set the lower bound threshold for the posterior
probability of a token to be accepted. This threshold is used by the
TypicalAcceptanceSampler to make sampling decisions during speculative
decoding. Defaults to 0.09
  --typical-acceptance-sampler-posterior-alpha
TYPICAL_ACCEPTANCE_SAMPLER_POSTERIOR_ALPHA
                        A scaling factor for the entropy-based threshold
for token acceptance in the TypicalAcceptanceSampler. Typically defaults
to sqrt of --typical-acceptance-sampler-posterior-threshold i.e. 0.3
  --disable-logprobs-during-spec-decoding
[DISABLE_LOGPROBS_DURING_SPEC_DECODING]
                        If set to True, token log probabilities are not
returned during speculative decoding. If set to False, log probabilities
are returned according to the settings in SamplingParams. If not
specified, it defaults to True.
                        Disabling log probabilities during speculative
decoding reduces latency by skipping logprob calculation in proposal
sampling, target sampling, and after accepted tokens are determined.
  --model-loader-extra-config MODEL_LOADER_EXTRA_CONFIG
                        Extra config for model loader. This will be
passed to the model loader corresponding to the chosen load_format. This
should be a JSON string that will be parsed into a dictionary.
  --ignore-patterns IGNORE_PATTERNS
                        The pattern(s) to ignore when loading the
model.Default to 'original/**/*' to avoid repeated loading of llama's
checkpoints.
  --preemption-mode PREEMPTION_MODE
                        If 'recompute', the engine performs preemption
by recomputing; If 'swap', the engine performs preemption by block
swapping.
  --served-model-name SERVED_MODEL_NAME [SERVED_MODEL_NAME ...]
                        The model name(s) used in the API. If multiple
names are provided, the server will respond to any of the provided
names. The model name in the model field of a response will be the first
name in this list. If not
                        specified, the model name will be the same as
the --model argument. Noted that this name(s)will also be used in
model_name tag content of prometheus metrics, if multiple names
provided, metricstag will take the first
                        one.
```

```
  --qlora-adapter-name-or-path QLORA_ADAPTER_NAME_OR_PATH
                        Name or path of the QLoRA adapter.
  --otlp-traces-endpoint OTLP_TRACES_ENDPOINT
                        Target URL to which OpenTelemetry traces will be
sent.
  --collect-detailed-traces COLLECT_DETAILED_TRACES
                        Valid choices are model,worker,all. It makes
sense to set this only if --otlp-traces-endpoint is set. If set, it will
collect detailed traces for the specified modules. This involves use of
possibly costly and or blocking
                        operations and hence might have a performance
impact.
  --disable-async-output-proc
                        Disable async output processing. This may result
in lower performance.
  --override-neuron-config OVERRIDE_NEURON_CONFIG
                        override or set neuron device configuration.
  --lookahead-cache-config-dir LOOKAHEAD_CACHE_CONFIG_DIR
                        Folder path of lookahead cache config
  --cpu-decoding-memory-utilization CPU_DECODING_MEMORY_UTILIZATION
                        the memory is used for lookahead cache, which
can range from 0 to 1. If unspecified, will use the default value of
0.15.
  --cpu-prefill-memory-utilization CPU_PREFILL_MEMORY_UTILIZATION
                        the memory is used for prefill cache, which can
range from 0 to 1. If unspecified, will use the default value of 0.3.
  --ignore-prompt-for-lookahead-cache
                        If True, the prompt will be ignored.
  --enable-prefix-cache-offload
                        Enables prefix cache offloading
  --apc-offload-not-lazy
                        If set, lazy launch of layer 2~n-1 will be
disabled.
  --apc-offload-min-access-threshold APC_OFFLOAD_MIN_ACCESS_THRESHOLD
                        Min threshold for evict offloading. Default 1.
  --apc-offload-enable-hit-cnt
                        Enable hit count in APC.
  --apc-offload-gpu-evictor-limit APC_OFFLOAD_GPU_EVICTOR_LIMIT
                        The free table size limited in gpu evictor. -1
default disable.
  --disable-log-requests
                        Disable logging requests.
  --max-log-len MAX_LOG_LEN
```

```
                        Max number of prompt characters or prompt ID
numbers being printed in log. Default: Unlimited
```

除了兼容 vLLM 所有的配置参数外，TACO-LLM 还额外添加了以下参数配置：

```
# Lookahead-Cache
--lookahead-cache-config-dir LOOKAHEAD_CACHE_CONFIG_DIR
                        Folder path of lookahead cache config
--ignore-prompt-for-lookahead-cache
                        If True, the prompt will be ignored.
--cpu-decoding-memory-utilization CPU_DECODING_MEMORY_UTILIZATION
                        the memory is used for lookahead cache, which
can range from 0 to 1. If unspecified, will use the default value of
0.15.

# Auto Prefix Cache CPU Offload
--enable-prefix-cache-offload
                        Enables prefix cache offloading
--cpu-prefill-memory-utilization CPU_PREFILL_MEMORY_UTILIZATION
                        the memory is used for prefill cache, which can
range from 0 to 1. If unspecified, will use the default value of 0.3.
--apc-offload-not-lazy
                        If set, lazy launch of layer 2~n-1 will be
disabled.
```

Sampling API

最近更新时间：2024-12-27 21:30:43

```
class taco_llm.SamplingParams(
    n: int = 1,
    best_of: Optional[int] = None,
    presence_penalty: float = 0.0,
    frequency_penalty: float = 0.0,
    repetition_penalty: float = 1.0,
    temperature: float = 1.0,
    top_p: float = 1.0,
    top_k: int = -1,
    min_p: float = 0.0,
    seed: Optional[int] = None,
    use_beam_search: bool = False,
    length_penalty: float = 1.0,
    early_stopping: Union[bool, str] = False,
    stop: Optional[Union[str, List[str]]] = None,
    stop_token_ids: Optional[List[int]] = None,
    ignore_eos: bool = False,
    max_tokens: Optional[int] = 16,
    min_tokens: int = 0,
    logprobs: Optional[int] = None,
    prompt_logprobs: Optional[int] = None,
    detokenize: bool = True,
    skip_special_tokens: bool = True,
    spaces_between_special_tokens: bool = True,
    logits_processors: Optional[Any] = None,
    include_stop_str_in_output: bool = False,
    truncate_prompt_tokens: Optional[Annotated[int, msgspec.Meta(ge=1)]]
= None,
    no_repeat_ngram_size: int = 0
)
    """Sampling parameters for text generation.

    Overall, we follow the sampling parameters from the OpenAI text
completion
    API (https://platform.openai.com/docs/api-
reference/completions/create).
    In addition, we support beam search, which is not supported by
OpenAI.
```

```
    Args:
        n: Number of output sequences to return for the given prompt.
        best_of: Number of output sequences that are generated from the
prompt.
            From these `best_of` sequences, the top `n` sequences are
returned.
            `best_of` must be greater than or equal to `n`. This is
treated as
            the beam width when `use_beam_search` is True. By default,
`best_of`
            is set to `n`.
        presence_penalty: Float that penalizes new tokens based on
whether they
            appear in the generated text so far. Values > 0 encourage
the model
            to use new tokens, while values < 0 encourage the model to
repeat
            tokens.
        frequency_penalty: Float that penalizes new tokens based on
their
            frequency in the generated text so far. Values > 0 encourage
the
            model to use new tokens, while values < 0 encourage the
model to
            repeat tokens.
        repetition_penalty: Float that penalizes new tokens based on
whether
            they appear in the prompt and the generated text so far.
Values > 1
            encourage the model to use new tokens, while values < 1
encourage
            the model to repeat tokens.
        temperature: Float that controls the randomness of the sampling.
Lower
            values make the model more deterministic, while higher
values make
            the model more random. Zero means greedy sampling.
        top_p: Float that controls the cumulative probability of the top
tokens
            to consider. Must be in (0, 1]. Set to 1 to consider all
tokens.
```

```
        top_k: Integer that controls the number of top tokens to
consider. Set
            to -1 to consider all tokens.
        min_p: Float that represents the minimum probability for a token
to be
            considered, relative to the probability of the most likely
token.
            Must be in [0, 1]. Set to 0 to disable this.
        seed: Random seed to use for the generation.
        use_beam_search: Whether to use beam search instead of sampling.
        length_penalty: Float that penalizes sequences based on their
length.
            Used in beam search.
        early_stopping: Controls the stopping condition for beam search.
It
            accepts the following values: `True`, where the generation
stops as
            soon as there are `best_of` complete candidates; `False`,
where an
            heuristic is applied and the generation stops when is it
very
            unlikely to find better candidates; `"never"`, where the
beam search
            procedure only stops when there cannot be better candidates
            (canonical beam search algorithm).
        stop: List of strings that stop the generation when they are
generated.
            The returned output will not contain the stop strings.
        stop_token_ids: List of tokens that stop the generation when
they are
            generated. The returned output will contain the stop tokens
unless
            the stop tokens are special tokens.
        include_stop_str_in_output: Whether to include the stop strings
in
            output text. Defaults to False.
        ignore_eos: Whether to ignore the EOS token and continue
generating
            tokens after the EOS token is generated.
        max_tokens: Maximum number of tokens to generate per output
sequence.
        min_tokens: Minimum number of tokens to generate per output
sequence
```

```
            before EOS or stop_token_ids can be generated
        logprobs: Number of log probabilities to return per output
token.
            When set to None, no probability is returned. If set to a
non-None
            value, the result includes the log probabilities of the
specified
            number of most likely tokens, as well as the chosen tokens.
            Note that the implementation follows the OpenAI API: The API
will
            always return the log probability of the sampled token, so
there
            may be up to `logprobs+1` elements in the response.
        prompt_logprobs: Number of log probabilities to return per
prompt token.
        detokenize: Whether to detokenize the output. Defaults to True.
        skip_special_tokens: Whether to skip special tokens in the
output.
        spaces_between_special_tokens: Whether to add spaces between
special
            tokens in the output.  Defaults to True.
        logits_processors: List of functions that modify logits based on
            previously generated tokens, and optionally prompt tokens as
            a first argument.
        truncate_prompt_tokens: If set to an integer k, will use only
the last k
            tokens from the prompt (i.e., left truncation). Defaults to
None
            (i.e., no truncation).
        no_repeat_ngram_size:
            If set to int > 0, all ngrams of that size can only occur
once.
    """
```

除了兼容 vLLM 所有的采样参数外，TACO-LLM 还额外添加了以下采样参数：

```
no_repeat_ngram_size:
            If set to int > 0, all ngrams of that size can only occur
once.
```

TACO LLM 性能

最近更新时间：2025-04-17 15:08:12

下载性能测试包

性能测试可以参见我们提供的性能测试包。

```
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/customers/common/benchmarks/taco_llm_demo.tar.gz
```

解压

```
tar -zxvf taco_llm_demo.tar.gz
cd taco_llm_demo
```

初始化环境

下载数据集、参考模型等。

```
#!/bin/bash

# 需要在有外网的环境下，先下载对应的测评需要用到的数据集 如果本目录有了，则忽略
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/llm/data/ShareGPT_V3_unfiltered_cleaned_split.json
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/llm/data/c4_sample.jsonl
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/llm/data/medical_dialogue.json
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/llm/data/github_sample.jsonl


# 下载参考模型Llama-2-7b-hf，注意更多模型可以在huggingface下载或者找我们接口人看
cos是否已经有
wget -c https://taco-1251783334.cos.ap-
shanghai.myqcloud.com/llm/llama/llama-2/Llama-2-7b-hf.tar
```

> ⚠ **注意：**
> 如果客户自己有数据集也可以在相关的脚本中调整、如果有相同的数据集，也可以不用再下载。

在线场景性能测试

运行 server 端

```
bash taco_bench_server.sh taco_llm
```

可以直接执行 server 端脚本，后面附加一个框架名称，例如 taco_llm。启动该脚本的目的是创建一个server端的等待任务，待 client 请求处理。
server 脚本中关键参数：大多数参数可参照本文中的在线模型进行配置。以下是更多参数配置的说明：

```
chat_template="./llama.jinja"          # chat配置的模板路径，包里已经包含

# 设置prompt参数
SYSTEM_PROMPT_LENGTH=0
tgt_max_len=300                        # 请求生成的最大长度
NUM_PROMPT_PRE_TGT=5                   # 每个并发请求数
NUM_TGT=1                              # 每秒并发数


#设置服务器参数
host="127.0.0.1"                       # 服务器地址
port="8007"                            # 服务端口
max_num_batched_tokens=10240           # 表示每次执行推理时支持最长的处理token
数，多个batch一次处理的token总数。
max_num_seqs=32                        # 后端支持最大的batch数
```

运行 client 端

```
bash taco_bench_client.sh taco_llm
```

client 脚本中关键参数：大多数参数可以参考本文中提到的在线模型。更多参数配置的说明如下：

```
DATASET_PATH="./ShareGPT_V3_unfiltered_cleaned_split.json"        # 数据集
路径
MODEL_PATH="/models/Llama-2-7b-hf"                                # 模型路
径
tp=1                                                              # 需要的
GPU卡数量
TOKENIZER_PATH=$MODEL_PATH

# 设置prompt参数
```

```
SYSTEM_PROMPT_LENGTH=0
tgt_max_len=300                          # 请求生成的最大长度
NUM_PROMPT_PRE_TGT=5                     # 每个并发请求数
NUM_TGT=1                                # 每秒并发数

# 设置输出长度
output_len=100                           # 设置每个请求最大输出数量

#设置服务器参数                          # 服务端地址，需要和server配套
backend=${1}
host="127.0.0.1"
port="8007"

ENABLE_PREFIX_CACHE=true                 # true/false: 打开/关闭 Auto Prefix
Cache功能
ENABLE_CACHE_OFFLOAD=true                # true/false: 打开/关闭 Cache Offload
功能
ENABLE_HIT_CNT=true                      # true/false: Cache命中情况打印
ENABLE_LOOKAHEAD=false                   # true/false: 打开/关闭 lookahead功能
```

结果

我们将结果根据相关指标存储在本地的 results 目录中。

```
============ Serving Benchmark Result ============
Backend:                                 taco_llm
Traffic request rate:                    inf
Successful requests:                     4
Benchmark duration (s):                  9.58
Total input tokens:                      1980
Total generated tokens:                  1600
Total generated tokens (retokenized):    1602
Request throughput (req/s):              0.42
Input token throughput (tok/s):          206.76
Output token throughput (tok/s):         167.08
----------------End-to-End Latency----------------
Mean E2E Latency (ms):                   2390.31
Median E2E Latency (ms):                 2069.78
---------------Time to First Token----------------
Mean TTFT (ms):                          656.38
Median TTFT (ms):                        41.38
P99 TTFT (ms):                           2427.64
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          4.35
Median TPOT (ms):                        4.51
P99 TPOT (ms):                           5.50
---------------Inter-token Latency----------------
Mean ITL (ms):                           12.43
Median ITL (ms):                         12.38
P99 ITL (ms):                            13.27
==================================================
```

```
taco_llm_demo/results/**.csv
```

csv 表格结果如下：

Name	Backend	e prefix	cache o	ble hit	le looka	rk durat	input t	enerated	throughp	n through	2E Latenc	2E Laten	n TTFT (m	an TTFT	9 TTFT (m	n TPOT (m	an TPOT	9 TPOT (m	an ITL (m	ian ITL	9 ITL (ms)
20241220	taco_llm	false	false	false	true	9.576429	1980	1600	206.7577	167.0769	2390.311	2069.776	656.3841	41.38155	2427.637	4.345681	4.508368	5.503256	12.4275	12.37992	13.26822

一键测试

客户也可以使用一个脚本完成 server/client 端测评部署并直接得到结果

```
bash taco_bench.sh taco_llm
```

> ⚠ **注意：**
> 实际上，这个脚本将上述的 server 端和 client 端两个脚本合并，在启动 server 端后等待10秒，然后启动 client 端。