大型语言模型推理的能量考量及效率优化

Jared Fernandez[*1], **Clara Na**[*1], **Vashisth Tiwari**[*1],
Yonatan Bisk[1], **Sasha Luccioni**[2], **Emma Strubell**[1]

[1]Carnegie Mellon University, [2]Hugging Face,
Correspondence: { jaredfern, clarana, vashisthtiwari } @cmu.edu

Abstract

随着大型语言模型（LLM）的规模和应用的扩大，其计算和环境成本也在不断上升。先前的基准测试工作主要集中在理想化环境中减少延迟，往往忽视了影响能源使用的多样化现实世界推理工作负载。在这项工作中，我们系统地分析了常见推理效率优化在各种自然语言处理（NLP）和生成式人工智能（AI）工作负载（包括会话 AI 和代码生成）中的能源影响。我们引入了一种建模方法，通过对输入输出令牌分布和批量大小的变化进行分箱策略来逼近现实世界的大型语言模型工作流。我们的实证分析涵盖了软件框架、解码策略、GPU 架构、在线和离线服务环境及模型并行配置。我们显示推理优化的有效性对工作负载几何、软件栈和硬件加速器高度敏感，表明基于 FLOPs 或理论 GPU 利用率的简单能源估计显著低估了真实世界中的能源消耗。我们的研究结果揭示，恰当地应用相关推理效率优化可以使总能源消耗比未优化的基线减少最多 73 %。这些见解为可持续的大型语言模型部署提供了基础，并为未来 AI 基础设施的节能设计策略提供了信息。

1 引言

大型语言模型（LLM）在任务性能上的提升促使人们对计算硬件和能源基础设施进行大规模投资，以支持 LLM 和相关机器学习模型的开发和部署 (**???**) 。然而，LLM 的日益普及导致其开发和部署所需的能源需求、水使用量和碳排放量相应增加 (**????**) 。主要受 LLM 和 AI 工作负载增加需求的推动，预测估计到 2030 年数据中心将消耗美国总能源需求的 9.1 % 到 11.7 % (**???**) 。然而，这种能源使用的预测主要依赖于行业范围的需求估计或对个别模型的能源需求的重大简化。

为了为这种日益增长的需求制定有效的能源政策，有必要表征开发（即模型训练）和部署（即推理）的基本计算工作负载。特别是，由于模型在重复使用中服务的规模和频率增加，推理的成本和效率尤为关键。具体而言，Meta 报告称，推理工作负载构成了其 AI 电力消耗的最多 70% ，而 Google 则将其机器学习能源归因于 60% 以及机器学习 AWS 云计算需求的 80% 到 90% 之间。

为了解决推理效率问题，自然语言处理和机器学习研究界开发了多种优化措施，涵盖了算法、软件框架和硬件加速器。这些优化主要针对模型速度的提升（例如延迟和吞吐量；(**??**) ）。此外，这些方法常常在受限环境或简化数据集上进行评估，这些环境和数据集无法反映真实任务的广泛多样性。这些任务范围从传统的自然语言处理应用，如序列标注和摘要，到更具计算需求的工作负载，如合成数据生成和链式思维推理。特别是当在现实环境中共同应用效率干预措施时，对语言模型推理的能源成本的理解仍然存在一个关键缺口。

在这项工作中，我们研究了大型语言模型（LLM）推理的能耗，并对以下方面的影响进行了全面分析：数据维度、解码策略、服务框架、编译技术、GPU 硬件平台、模型并行性以及架构变体对推理过程中的总能耗的影响。基于我们对这些优化的能耗分析，我们根据实际工作负载中可变序列长度和批处理的情况，近似计算了 LLM 离线推理的能耗，考虑了未经优化的推理的上限和理论优化推理的下限。我们的分析表明，尽管理想化的硬件利用率估算大大低估了语言模型推理的能耗，但通过正确应用推理效率优化，可以将推理的能耗在未经优化的基线（使用原始的 PyTorch 和 Huggingface Transformers）上减少至多 73 %，接近于理论理想性能 26.6 % 的差距（见表 4 ）。

在下节中，我们将描述用于评估推理效率的实验设置。

我们将实验重点放在参数数量从 1B 到 32B 的语言模型上，主要评估 `Llama-3.1-8B-Base` 和 `Llama-3.1-8B-Instruct` 模型作为代表性的仅解码 transformer (**?**) 。为了研究模型架构扩展的效果，我们纳入了 `Qwen-1.5-32B` 模型 (**?**) 。在架构比较中，我们分析了稀疏的 `OLMoE` 专家混合（MoE）模型及其密集对应物——1B 和

[*]Equal contribution

7B OLMo 架构——它们分别维持了可比的活跃参数和总参数数量 (**??**) 。

数据维度 我们研究了数据维度在三个关键维度上的影响：输入序列长度、输出生成长度和批量大小。

使用大型语言模型进行推理通常分为两个阶段：预填充和标记生成，每个阶段具有不同的能量特征 (**?**) 。预填充阶段并行处理提示，通常受计算限制，实现高 GPU 利用率。相反，自回归解码阶段通常受内存限制，导致 GPU 未充分利用。这些瓶颈及其导致的能量特征会随着输入和输出长度的变化而变化。

为了应对生成过程中 GPU 的低效利用，服务系统采用批处理策略。然而，批处理的效果随输入输出特性 (**??**) 的变化而不同。由于内存限制，较长的输入序列限制了最大批次大小，而可变的输出长度可能导致批次利用效率低下，因为某些序列比其他序列提前完成。

我们的分析涵盖了从 1（单样本推理）到任务相关的最大值（最多 1024）的批量大小，确保覆盖范围包括最大化的现实设置。

我们基于涵盖文本分类、总结、翻译和开放式文本生成的 NLP 工作负载进行分析。不同的任务表现出不同的数据维度：分类涉及最小的生成（通常是单个标记），总结将长上下文与中等长度的输出配对，而翻译通常假定输入输出长度是平衡的。表 3 展示了所考虑数据集中的输入长度统计数据。

在一个受控的扫描中，我们探索了输入令牌最多达到 32k 和输出令牌达到 4k 的场景，通过以二的幂次来改变序列长度。当改变上下文长度时，我们将生成固定为 64 或 8 个令牌，并假设输入上下文为 512 或 64 个令牌来改变输出长度。输入上下文长度通过截断来自 PG19 (**?**) 的更长序列来维持。

解码策略。 用于生成的不同解码策略具有不同的计算特征，并可能对生成效率产生重大影响 (**?**) 。为了研究采样方法和自回归解码策略的影响，我们研究贪心解码、束搜索解码、温度采样、top- p 解码对能量需求和端到端延迟的影响 (**?**) 。

除了自回归解码，我们还研究了推测解码的影响。推测解码通常被用作一种延迟最小化的推理优化 (**?**) 。在推测解码中，一个轻量级的草稿模型用于预测多个标记 (γ)，然后这些标记由目标模型并行验证 (**??**) 。通过更好地利用 GPU，相较于自回归解码，推测解码提供了延迟的改善。

在我们的实验中，我们使用以下目标草稿模型对，并在各种批量大小中采用超前值 $\gamma = 4$ ：`DeepSeek-R1-Distill-Qwen-32B` 与 `mobiuslab sgmbh/DeepSeek-R1-ReDistill-Qwen-1.5B-v1.1` (**??**) ； `Llama-3.1-8B-Base` 与 `Llama-3.2-1B` (**?**) 。

软件优化。 在推理使用的软件框架选择显著影响延迟和能源效率，这是通过优化的内核实现和计算图管理来实现的。(**??**) 我们评估了两种广泛采用的 LLM 推理库：原生 PyTorch 与 HuggingFace transformers (**?**) ，以及 vLLM, 一个优化的 LLM 推理框架，可实现更好的计算和内存利用率。(**??**) 实验在 `bfloat16` 精度下进行。

在这些框架内，我们与通过 TorchInductor 实现即时编译的原生 PyTorch 基准（即 `torch.compile` ）和 CUDA 图形内核序列化进行比较。此外，对于 vLLM，我们评估了连续批处理，它通过叠加序列 (**?**) 有效处理批处理中可变的输出长度。

硬件平台。 我们的实验是在一个有多个 GPU 类型和节点配置的内部异构服务器上进行的。具体来说，我们在多个代的消费级工作站和数据中心 GPU 加速器上进行实验，这些加速器属于安培（A6000，A100 80GB PCIe）和阿达·洛夫莱斯（A6000 Ada）微架构。

所有实验均在 8-GPU 节点上运行，并为每种 GPU 类型配置标准化的节点和作业级 CPU 和 RAM 设置。对于多 GPU 实验，我们最多同时使用 4 个 GPU，研究设备组大小为 2 和 4 的张量并行推理。[1] 。我们使用 `Llama-3.1-8B` 和 `Qwen-32B` 模型，检查标准和预测性解码方法。关于计算硬件的附加细节在附录 **??** 中提供。

我们通过测量对 1,024 个样本进行推理所需的延迟、吞吐量、GPU 能量和 GPU 功率来评估推理的效率。总能耗和 GPU 功率指标是通过使用 `CodeCarbon` 库的 Nvidia Management Library (NVML) 进行测量的。在评估之前，我们会对多达 20 个批次进行预热，以便进行内存分配、所需的 CUDA 图捕获和 JiT 编译。结果以三次运行的平均能耗、延迟或功率使用值来报告。

在接下来的部分中，我们将研究数据维度变化、模型架构、解码策略以及软件优化对推理能耗的影响。

1.1 数据集和序列长度的影响

我们在图 1 中展示了我们对序列长度和批量大小进行控制扫过的结果。当序列长度超过 128

[1] 该配置为其他用户留出了 4-7 个 GPU。虽然 Slurm 调度程序在网络、内存和 CPU 基础设施方面不强制对任务进行完全隔离，但在实践中，并发工作负载在 CPU 或内存方面都不足以对我们的任务造成重大影响——例如，在绝大多数情况下（98 %），在我们作业运行的节点上进行的 RAM 利用率测量小于总可用内存的 20% 。

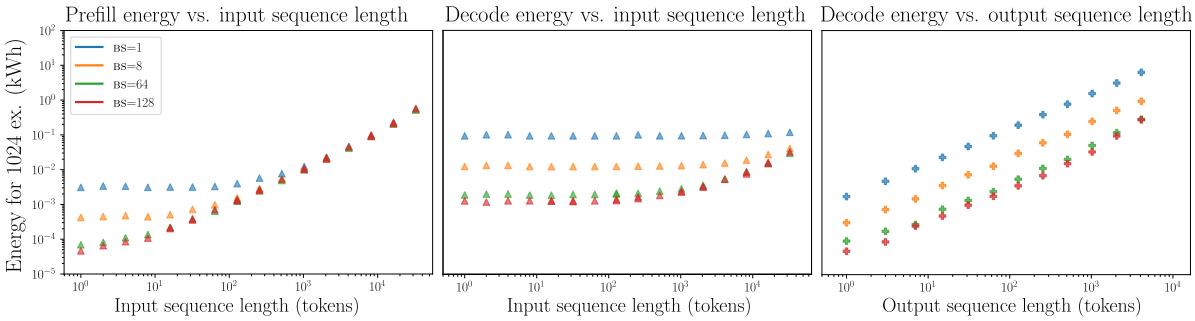


Figure 1: 在 A6000 GPU 上对输入和输出序列长度进行受控扫描，使用的 vLLM 后端在 §1.1 中描述。我们将推理成本分解为预填充和解码能耗。在小批量大小和输入序列长度的情况下，工作负载的能量强度随序列长度的增加以次线性增长。每个标记的解码比预填充更耗能，但在短生成和小批量大小情况下，即使使用 vLLM 框架，能量强度也开始线性增长。

个标记时，预填充成本随着输入序列长度的增加而增加，无论批量大小如何，其增长速率是相同的。在较短的序列长度和较小的批量大小时，由于加速器显著未达到饱和状态，预填充的能耗是恒定的，且与计算工作量无关。虽然我们将输出生成标记固定为 64，但我们验证了当固定生成长度为 8 个标记时，这种能量强度增加率的收敛发生在同一点；见附录 **??** 中的图 9 。

在图 1 中，解码的能量强度也仅在较大的输入序列长度时与输入上下文长度缩放。

然而，由于解码的自回归和顺序性质，解码的能量强度与序列长度线性扩展，无论序列长度或批量大小如何。

一般来说，在所有情况下，解码能量主导着整体工作负载，除了那些具有最短生成长度的情况，比如分类工作负载和短篇摘要中出现的情况。请注意对数-对数刻度和平行的线性趋势，其中截距的差异与批量大小 [2] 的差异成比例。在接下来的章节中，我们将讨论适合不同类型工作负载几何的多种算法和软件干预措施。

1.2 算法优化的影响

预测性解码仅在较小批次时降低能耗。 在低批量推理中，推测性解码常用于实现推理提速，此时自回归解码无法实现高 GPU VRAM 利用率。然而，对于已达到 GPU 饱和的大批量，草稿模型的推测和过度验证引入了额外的开销。在大批量情况下，对于短到中等上下文，LLM 推理通常受限于计算，这使得推测性解码比目标模型 (**??**) 自回归解码更慢。

与替代解码策略和采样方法导致的能耗变化相比，推测性解码对语言模型推理的能耗和延迟有最大的影响。在较小的批处理大小（≤ 16

[2] 关于原生 PyTorch 后端的更多结果，请参见附录 **??** 中的图 8 ，以及图 10 比较经典 NLP 任务样本的实际能量强度测量。

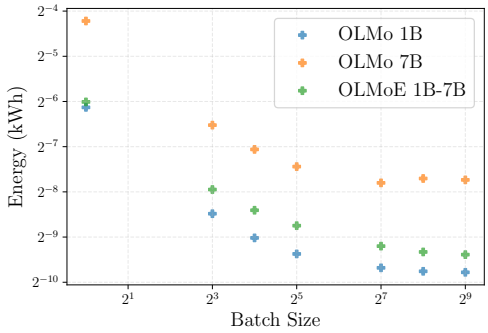
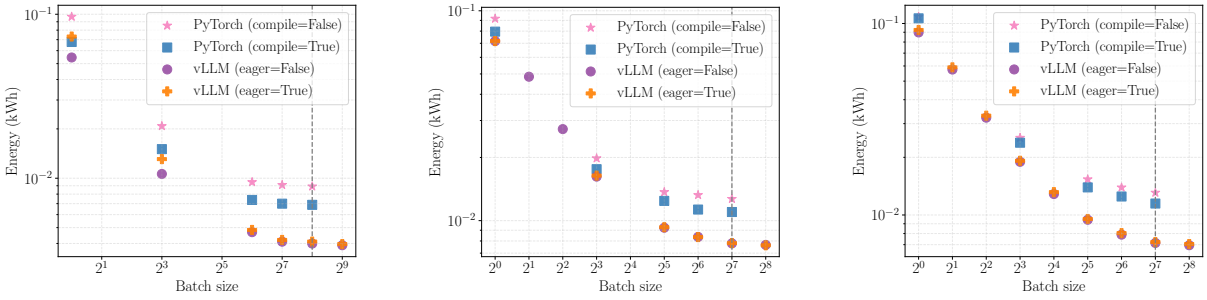


Figure 2: 专家混合大型语言模型（LLM）比具有可比活动参数的密集模型需要更多的能量；这种差异在更大的批处理大小下更为明显。

）下，推测性解码在减少推理的总能耗方面效果显著，与单个示例推理相比减少了最多 +29.14%（图 **??** ）。然而，在较大的批处理大小下，自回归解码方法更为高效，当批处理大小为 128 时，推测性解码需要多 25.65% 的能量。

专家混合导致更高的推理能耗。 稀疏专家混合模型通常被用作替代架构，因为在训练过程中它们的样本效率更高，并且相对于具有相同激活参数数量的密集神经网络，其性能有所提升。尽管密集型 `OLMo-1B` 和 `OLMoE1B-7B` 专家混合模型的能耗远低于密集型 `OLMo-7B` 模型，但 OLMoE 架构的能耗比基础 OLMo 1B 模型高达 54.24% ，尽管它们具有相似数量的激活参数。

我们发现，MoE 的能量和延迟增加可以归因于专家层使用的融合内核，其运行速度明显慢于稠密模型中线性层的相应 GEMM 操作；在批量大小为 1 时，运行速度慢了 19.70 %，在批量大小为 8 时，慢了 63 %。值得注意的是，我们观察到，MoE 模型中的额外路由操作引入了极小的延迟；并且更多 CUDA 图和内核启动操作的增加开销，通过内核序列化和图编译优化（即使用 CUDA 图的 vLLM）在很大程度



(a) A100 80GB PCIe (b) A6000 Ada (c) A6000

Figure 3: 在 PyTorch 和 vLLM 后台中针对 64 个输出标记的 1024 个样本进行推理时，不同 GPU 的能耗比较。对于每个 GPU，我们分别比较了使用和不使用编译的 PyTorch，以及 vLLM 使用和不使用 CUDA Graph 序列化的情况。黑色线条代表 PyTorch 允许的最大批处理大小。相对节省在低批处理大小范围内最为明显，而由于 vLLM 的优化，它可以处理更大的批处理大小。

上得到了缓解。

1.3 软件优化的效果

与原生 PyTorch 相比，vLLM 推理服务引擎提高了吞吐量和能源效率。vLLM 框架使用 PagedAttention 来实现非连续的 KV 缓存块，这减少了在稀疏序列情况下的内存碎片化和冗余内存分配 (**?**) 。

这些优化提高了内存效率，并使 vLLM 框架能够在固定内存的 GPU 上支持更大的批处理大小。

编译和内核序列化提高了效率。 图形编译和内核序列化通过消除计算图中的冗余操作并分别减少内核启动开销来增加硬件利用率。我们观察到，`torch.compile` 和 CUDA 图形序列化（`eager=False`）在不产生额外能耗的情况下提高了吞吐量，如图 3 所示。然而，我们注意到，CUDA 图形的优势在较小的批处理大小时更为明显，因为对于较小的计算工作负载，内核启动的相对成本更大。

连续批处理减少能源使用。 LLM 推理本质上是自回归的，需要许多连续的操作。静态批处理在整个推理过程中保持固定的批处理大小，这在生成长度变化且在提前终止后空闲计算累积时导致 GPU 未被充分利用。连续批处理通过动态地用新请求替换已完成的请求来缓解这一问题，提高了 GPU 的利用率并减少了空闲时间 (**?**) 。这种方法在生成长度具有高方差时特别有效，在较大批处理大小下实现显著的加速。

我们观察到，在较小的批量下，在线调度的开销超过了其好处，但在较大的批量下，连续批处理的在线服务所需的能量更少；详细信息见附录 **??** 。我们注意到，鉴于样本来自同一数据集，连续批处理的影响被低估了，从而减少了输入和输出长度的变化。

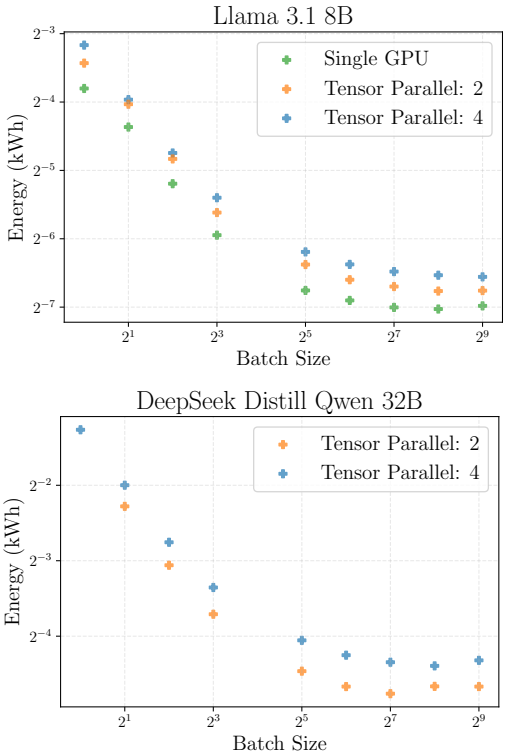


Figure 4: Llama-3.1 8B 和 Qwen 32B 在不同程度的张量并行性下的能量消耗。

1.4 硬件设计选择的影响

多 GPU 张量并行性减少了延迟以增加功耗 模型并行技术，如张量并行和流水线并行，通常用于缓解大量模型参数和批量大小带来的内存压力，并利用多个硬件加速器以加快工作负载的执行 (**?**) 。此外，对于固定的工作负载，张量并行在工作负载被分割到各个加速器上的同时，减少了每个设备的计算密度和功耗。但是，增加加速器所带来的加速效果不足以抵消更多设备使用的能量成本（即使用双倍的 GPU 却无法实现双倍的加速）。

在图 4 中，我们观察到使用张量并行从单个 GPU 推理扩展到四个 GPU 可以减少 Llama-3.1 8B 模型的延迟和每设备的功耗。然而，增加并行性由于加速器数量的增加导致更高的总能量使用。具体而言，将固定工作负载并行化到两个和四个 GPU 上会分别减少 40.16 % 和 61.34 % 的延迟，但由于增加了额外的设备，在单批次推理中总能量使用分别增加了 29.3 % 和 55.23 %。

硬件速度的影响 优化技术的有效性在不同硬件平台上差异显著，对于计算效率优化的加速器，速度更快的加速器表现出更大的收益。我们的结果表明，图编译、核序列化和预测解码在 A100 GPU 上达到了最大效果。

具体来说，PyTorch 编译在 A100 上带来了 29.90 % 的提升，而在 RTX 6000 Ada 上降至 13.28 %，在 A6000 上进一步降至 1.96 %。同样，vLLM 的 eager 模式优化在 A100 上显示了 25.47 % 的提升，而在 A6000 上为 2.97 %。这一模式表明，随着硬件计算能力的提高，软件优化针对内核效率的相对影响变得更加明显。

2 优化对推理能耗的影响

在本节中，我们概述了如何建模大型语言模型（LLM）在合成和真实工作负载分布下的能耗。我们利用经典的自然语言处理任务和推理请求的数据集来估算不同执行环境中的能耗，包括在单个 A6000 GPU 上通过软件优化的 PyTorch-native 和 vLLM 后端。

2.1 使用离线服务建模能量需求

我们考虑在离线环境中处理数据集 $\mathcal{D} = \{R_1, R_2, \ldots, R_N\}$ 所需的能量，其中所有请求可以自由地批处理，同时每个请求 R_k 包括一个元组 (i_k, o_k) ，代表了输入标记长度 i_k 和输出生成长度 o_k 。

$$R_k = (i_k, o_k), \quad \forall k \in \{1, \ldots, N\}.$$

由于 i_k 和 o_k 在不同请求中变化显著，我们利用数据集统计信息——包括输入和输出长度的中位数和 99 百分位数（在 §2.2 中讨论）来指导我们的分箱策略。

分箱策略。 为了有效处理广泛的 (i_k, o_k) 值范围，我们为输入和输出长度定义了离散的区间集：

$$\begin{aligned} I_{\text{bins}} &= \{2^m \mid m \in \mathbb{N}, 4 \le m \le 13\} \\ &= \{32, 128, 256, 512, 1024, 2048, 4096, 8192\}, \\ O_{\text{bins}} &= \{2^n \mid n \in \mathbb{N}, 3 \le m \le 9\} \\ &= \{8, 16, 32, 64, 128, 256, 512\}. \end{aligned}$$

这些区间选择确保在现实的请求分布中有足够的覆盖。值得注意的是，我们排除了极长的输入请求（$> 8k$ 个标记）和超过 512 个标记的生成输出。

给定一个请求 $R = (i, o)$ ，我们将其映射到最接近的上限桶：

$$\begin{aligned} I^* &= \min\{I \in I_{\text{bins}} \mid I \ge i\}, \\ O^* &= \min\{O \in O_{\text{bins}} \mid O \ge o\}. \end{aligned}$$

我们将同一 (I^*, O^*) 桶中的请求分组为大小为 $B(I^*, O^*)$ 的批次，这是给定硬件和后端配置的最大允许批量大小。每个批次并行处理 $B(I^*, O^*)$ 请求，从而允许更高效的能量利用，这更能代表现实世界的推理设置。

考虑到我们的硬件配置和后台，我们收集了 $E_{\text{batch}}(I^*, O^*)$ 的估算值，$E_{\text{batch}}(I^*, O^*)$ 对应的是用于处理批量大小为 B 、输入提示长度为 $I*$ 、输出长度为 $O*$ 的请求所需的能量。

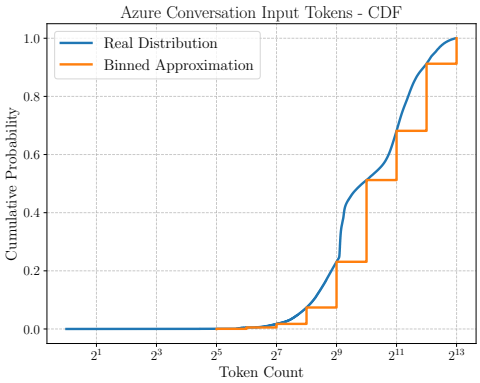
我们收集了实际的能量测量值 $\mathrm{E}^{\text{real}}_{\text{batch}}(\mathrm{I}^*, \mathrm{O}^*)$ ，表示在处理一个大小为 $B(I^*, O^*)$ 的完整批次时，输入长度为 I^* 和输出长度为 O^* 时观察到的能量使用。因此，为了处理分区中 N 请求的工作负载，总的估计能量消耗由以下公式表示：

$$\widehat{E}_{\text{total}} = \sum_{(I^*, O^*)} \left(\frac{N^{\text{real}}(I^*, O^*)}{B(I^*, O^*)} \right) E^{\text{real}}_{\text{batch}}(I^*, O^*),$$

，其中 $N^{\text{real}}(I^*, O^*)$ 是映射到分区 (I^*, O^*) 的观测请求总数，$\frac{N^{\text{real}}(I^*, O^*)}{B(I^*, O^*)}$ 表示处理它们所需的批次数。

作为一个简单的基线，我们通过制造商额定的硬件速度（$FLOPS_{HW}$ ）、功耗（TDP）和推理所需的浮点运算（FLOPs）来估计这些工作负载的能源效率上限 $FLOPs$ [3] 。该近似假定硬件在最大效率下被利用，包括在理想化的浮点运算吞吐量和最大功耗方面的利用。

[3] 基于 Nvidia 的数据表，RTX A6000 GPU，我们利用考虑 $FLOPS_{HW}$ 为 309.7 TFLOPS 和 300W TDP 功耗；并通过 DeepSpeed profiler (**?**) 估计理论推理 FLOPs。



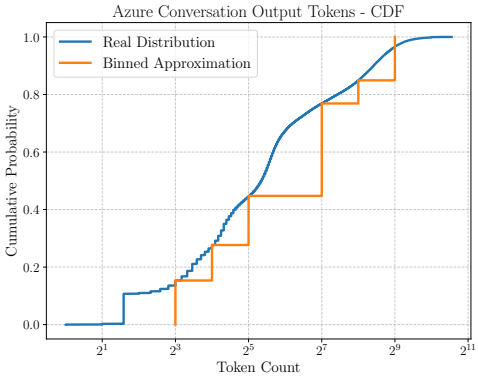


Figure 5: 对于 Azure 会话输入（左图）和输出（右图）令牌长度，比较真实令牌长度分布（蓝色）和分箱近似（橙色）。CDF 图说明了我们的分箱策略如何在确保能源估计计算效率的同时，近似经验分布。

$$\widehat{E}_{\text{Optimal}} = \left(\frac{\text{TDP}}{FLOPS_{HW}}\right) \times \sum_{(I^*,O^*)} N^{real}(I^*,O^*) \times FLOPs(I^*,O^*)$$

2.2 评估

Dataset	Mean ± Std	Median	99th
BurstGPT	256.80 ± 242.27	215	1038
Azure Chat	1631.58 ± 1529.64	928	6683
Azure Code	2511.28 ± 2133.54	1930	7685

Table 1: 输入序列长度在真实世界 LLM 工作负载中的统计

Dataset	Mean ± Std	Median	99th
BurstGPT	35.10 ± 108.59	7	478
Azure Chat	105.51 ± 158.25	41	694
Azure Code	22.69 ± 74.78	8	271

Table 2: 真实世界 LLM 工作负载中的输出序列长度统计

Task	Mean ± Std	Max	Output
Translation	49.96 ± 39.39	550	64
Generation	136.89 ± 93.13	547	64
Classification	292.48 ± 239.94	3112	1
Summarization	838.49 ± 400.70	2386	64

Table 3: 用于能量基准测试的 NLP 任务中的标记化输入和输出长度统计

我们研究了一组经典的自然语言处理任务和大语言模型推理工作负载，每个任务和工作负载都有不同的输入上下文和输出生成序列特征；数据集统计信息在表 3 、1 、2 中提供。我们模拟了一个在 RTX A6000 GPU 上的大规模离线处理场景，其中通过序列长度将示例进行分类处理（如 §2 中所述），并在 GPU 内存容纳的最大批量中并行处理。

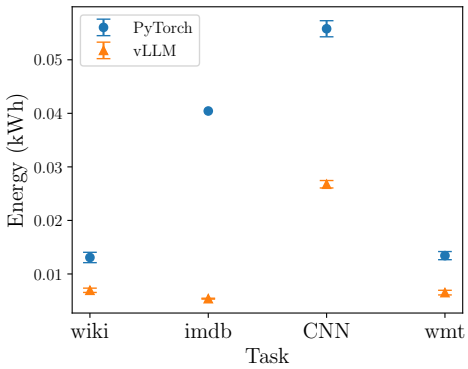


Figure 6: 能量比较，在对 1024 个样本进行推理时，PyTorch 关闭编译与 vLLM 关闭即时模型的对比。

利用第 2.1 节中描述的模拟工作负载，我们估计了第 2.1 节中评估的推理效率优化的有效性。基于这些结果，我们选择一个针对大批量推理的效率优化推理框架。具体来说，我们考虑在单个 GPU 上使用带有 CUDA 图序列化（关闭急切模式）的 vLLM 的密集模型进行推理，并将其与本机 PyTorch 的非优化推理作为能效下限进行比较。此外，我们还根据模型和硬件配置建立理想的能量基线模型。

经典自然语言处理任务。 我们在一组经典的自然语言处理任务中对英语的能量使用进行了基准测试：文本分类（IMDB，**?** ）、机器翻译（WMT-14，**?** ）、摘要（CNN-DailyMail，**?** ）和文本生成（Wikitext-2 (**?**) ）。

对于这些任务中的每一个，我们从数据集中抽取了 1024 个示例，输入和输出令牌的统计数据详见表 3 。我们注意到，输入序列被填充到最大序列长度。能量使用最少的最佳运行的能量配置概况总结在图 6 中，通过推理效率优化提供了一致的能量使用减少。

另外，我们还评估了在真实世界的大型语言模型（LLM）工作负载中能效优化的能源

强度和有效性。我们模拟了在应用于短对话的 Burst-GPT 数据集和长语境对话及代码补全的 Azure LLM 推理聊天和代码轨迹中使用的 LLM 推理请求的离线处理。每个数据集提供了 LLM 推理请求的轨迹及其对应的输入语境和输出生成长度。与传统的自然语言处理任务相比，现代 LLM 工作负载在输入语境和输出生成的标记长度上往往更长，其中代码辅助应用具有更长的语境，而对话场景则导致更长的生成。

Dataset	PyTorch % Δ	vLLM % Δ
BurstGPT	506.52 %	63.75 %
Azure Code	102.79 %	26.59 %
Azure Conversation	490.23 %	64.22 %

Table 4: 离线推理各种任务的能耗相对于理论值的百分比差异。

由于请求数量的增加和序列长度的加长，我们观察到这些工作负载需要大量的能量。然而，我们发现适当应用的推理效率优化可以显著降低能源成本，在 BurstGPT、Azure Code 和 Conversation 上分别节约 73.00 %、37.58 % 和 72.18 %。

3 相关工作

为了满足实际部署环境中的服务级别目标（SLO）服务需求，LLM 推理的效率优化通常旨在优化模型服务速度，该速度通过延迟和首次令牌响应时间来衡量。已经开发了多种方法来满足这些延迟限制，包括：连续批处理 (**?**)、模型并行性 (**???**)、猜测性解码 (**????**) 和解耦服务 (**?**)。

单纯为了速度优化系统性能是不够的，因为这样无法很好地表征模型推理的能耗和由此产生的碳排放；由于这样的优化方法可能需要额外的计算，或者在效率成本指标 (**?**) 之间表现出较低的相关性。最近的研究探讨了通过在异构硬件上分散服务 (**?**)、在系统范围内调度和请求路由到能源优化实例 (**?**) 以及通过提示指令诱导较短序列生成 (**?**) 等方法，明确减少 LLM 服务的能耗和碳排放。然而，针对延迟优化方法的能源需求的确切影响或改进尚未完全表征。

随着模型规模和部署普及性的增加，机器学习模型的能量消耗和碳排放问题在研究界和工业界引起了越来越多的关注。对大型语言模型（LLMs）能量需求和环境影响的估算主要集中在估算预训练和微调阶段的成本，因为模型开发所需的成本非常大；大型工业开发者同样报告了预训练所需的能量。

与训练相比，推断工作负载在请求频率、批处理、输入和输出序列长度上具有更高的可变性，并在规模上跨多种硬件平台执行；由于在生成的预填充和解码阶段的功率差异，其能量使用特征更加复杂。先前的工作研究了各种任务中机器学习模型的比较能耗、不同尺寸语言模型的能耗、硬件配置（如 GPU 功率限制和频率缩放）的影响以及序列长度的变化和批处理策略的影响。然而，这类推断能耗评估通常依赖于简化的部署设置，模型架构和服务框架的种类较少。

4 结论

在这项工作中，我们评估了常见的推理效率优化对大型语言模型服务的能量需求的影响。我们研究了多种优化技术，并对对应经典自然语言处理任务的数据和现代大型语言模型部署环境中的代表性数据进行了评估。我们得出结论，降低能耗的延迟优化有效性对输入数据的形状、底层硬件架构和软件框架实现非常敏感，并且不能统一应用优化。

此外，我们对经典的自然语言处理任务和真实世界的大型语言模型推理工作负载进行了案例研究，发现在 BurstGPT 聊天数据集上，恰当应用所研究的推理优化可以减少最多 73 % 的总能耗。

在这项工作中，我们评估了通过总 GPU 功耗近似的 LLM 推理的能效和碳排放。尽管 GPU 执行推理所需的大多数算术操作，并且其工作时的 TDP 高于其他组件，我们并不考虑硬件系统中其他组件的能源消耗，例如 CPU、内存或磁盘存储的功耗；也不估计其他硬件加速器架构（例如 TPUs，NPUs 等）的能源需求。同样的，我们对常用的推理软件框架和标准效率优化进行了研究。然而，还有其他设置和计算优化可以应用于 LLM 推理，例如：使用精度降低或混合精度、自适应调整 GPU 频率、额外形式的模型并行性或其他形式的负载管理和工作负载调度；这些均不在本工作的讨论范围之内。

在这项工作中，我们主要关注机器学习推理的运行能耗。推理的隐含成本估算以及机器学习训练的成本不在本工作的范围之内。

尽管对 LLM 推理的能耗进行改进的表征可以用来设计更高效的服务设置并减少推理的能量需求，但预训练成本的降低可能会导致更多的个人和组织进行大模型预训练（即杰文斯悖论）。

在表格 5 中，我们提供了用于我们基准测试实验的节点硬件配置的更多细节。

CPU	RAM	GPU	GPU TDP	FP32 TFLOPS	Bfloat16 TFLOPS
256xAMD EPYC 7763	1TB	Nvidia RTX A6000	300W	38.7	–
128xAMD EPYC 7513	500GB	Nvidia RTX A6000 Ada	300W	91.1	–
128xAMD EPYC 7763	1TB	Nvidia RTX A100-80 GB	300W	156	312

Table 5: 节点硬件规格

5 数据集许可

用于摘要的 CNN-DailyMail 数据集是根据 Apache-2.0 许可证发布的。用于文本生成的 Wikitext-2 数据集是依照知识共享署名-相同方式共享许可发布的。WMT-14 翻译数据集仅供非商业使用。BurstGPT 和 Azure 跟踪数据集是依据 CC-BY-4.0 许可证发布的。

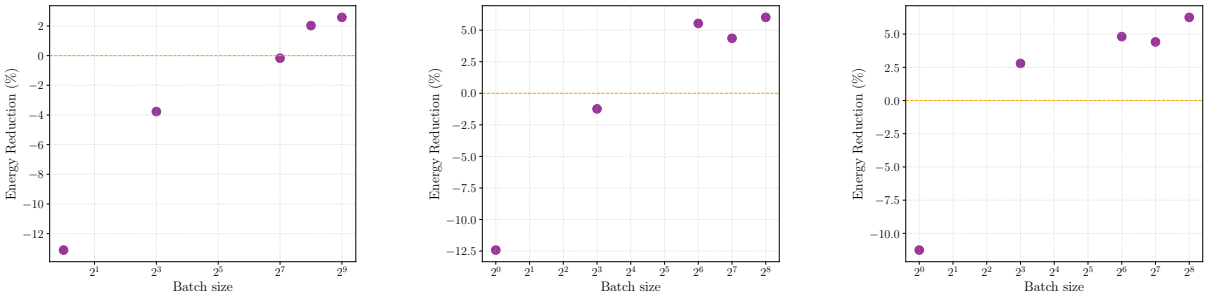
6 人工智能协助的确认

使用了人工智能辅助来帮助进行文献综述和代码完成辅助，特别是在创建可视化过程中。

在图 7 中，我们展示了有关 vLLM 连续批处理对在线推理影响的附加结果，我们观察到在大批量大小时，连续批处理减少了能量使用。

7 附加序列长度结果

在图 8 中，我们展示了使用 PyTorch 框架对输入和输出序列长度进行缩放的额外结果。



(a) A100 80GB PCIe (b) A6000 Ada (c) A6000

Figure 7: 跨不同 GPU 的在线和离线服务模式之间的能耗对比 $(E_{offline} - E_{online}) * 100/E_{offline})$。在线服务采用的优化在较大批量情况下节省多达 5 % 的能耗

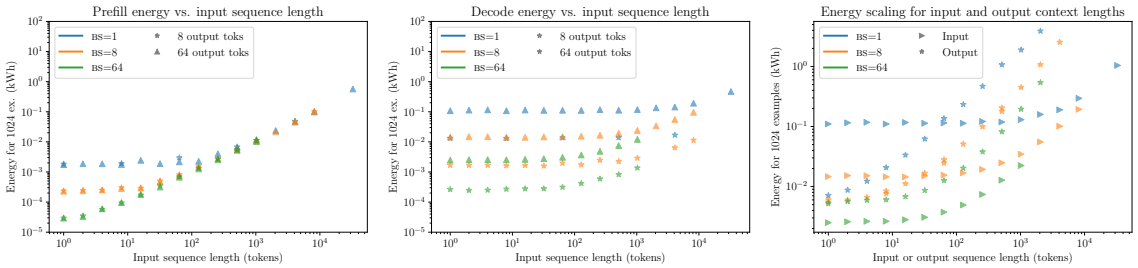


Figure 8: 在 A6000 GPU 上进行输入和输出序列长度的控制扫描，使用原生 PyTorch 后端。

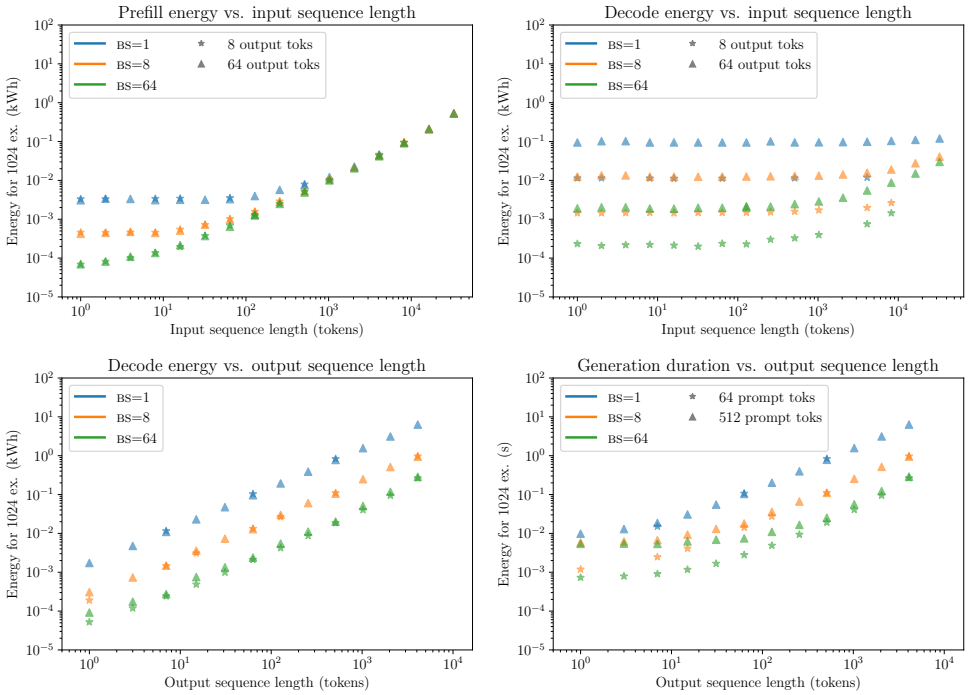


Figure 9: 在 A6000 GPU 上使用 vLLM 离线推理对输入和输出序列长度进行受控扫描。在这里，我们显示多个固定的序列长度大小以进行比较，同时在批量大小和序列长度的其他维度进行扫描。

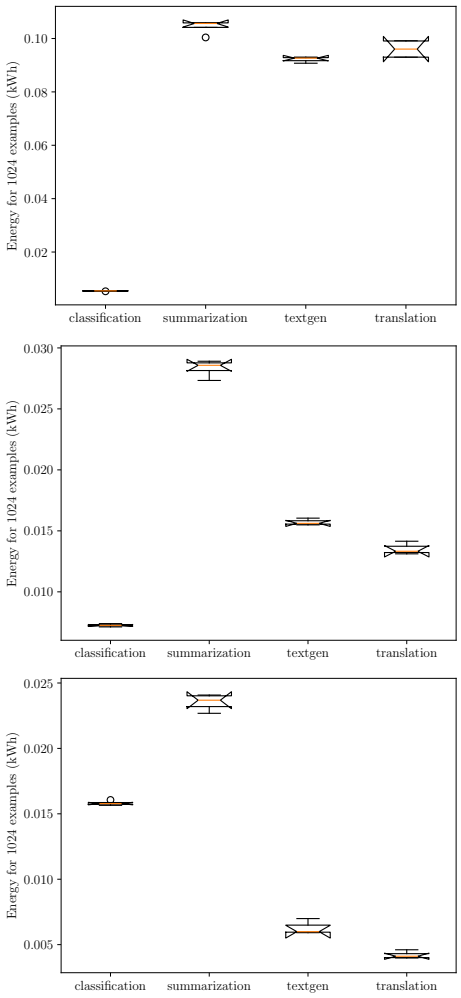


Figure 10: 使用 vLLM 后端的经典 NLP 任务及其能量强度。从上到下，批大小分别为 1、8 和 128