arXiv:2601.02298v1 [cs.CL] 5 Jan 2026

大语言模型（LLMs）中的二次幂量化感知训练（PoT-QAT）

Mahmoud Elgenedy斯坦福大学计算机科学系
melgened@stanford.edu

摘要

过去几年中，大语言模型（LLMs）的参数数量呈指数级增长，例如，从 GPT-2 的 15 亿个参数增加到 GPT-3 的 1750 亿个参数，更新的版本甚至可能超过万亿个参数。这给模型的实际部署带来了巨大挑战，尤其是在边缘设备上。与云计算不同，边缘设备的内存和处理能力十分有限，因此需要开发新的方法，使此类应用切实可行。本文研究了一种特殊的量化方法，通过将数值限制为二次幂（PoT）来压缩权重。这种方法只需存储指数，从而节省大量内存；更重要的是，它能以低成本的位移取代昂贵的乘法运算，大幅降低计算开销。为克服这种严格量化造成的性能损失，我们研究了量化感知训练（QAT），通过额外训练提升性能。GPT-2 124M 上的结果表明，经过额外训练后，量化二次幂模型的性能显著提升：困惑度改善了 66%，相较于基线 GPT-2 的 BERT-Score 损失为 1%。据估计，内存节省率为 87.5%，而与全精度相比，采用二次幂量化后的推理速度预计可提升 3 至 10 倍。

1 引言

量化是一种非常有效的方法，可用于优化深度神经网络（DNN）的复杂度；它通过降低数字的表示精度，帮助减少内存和计算需求 [1]。在本提案中，针对大语言模型，我们研究将权重限制为二次幂（PoT），以大幅减少内存占用和计算量。此外，为提升性能，我们在 [2] 的工作基础上，进一步探索量化感知训练（QAT）。有关采用 QAT 的其他大语言模型量化方法，请参阅 [3]。

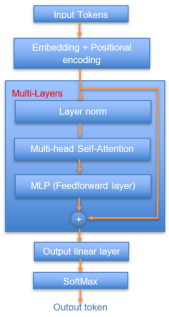
尽管神经网络量化已得到广泛研究 [4-6], ，但仅有少数研究探讨了二次幂量化 [7-10]。具体而言，一些研究针对通用深度神经网络探讨了将权重限制为二次幂 [7-8] ，也有研究侧重于使用卷积神经网络（CNNs）的图像应用 [9]。然而，对于大语言模型应用，相关研究非常有限，其中包括一篇近期发表的论文 [10] ，该论文提出采用训练后量化（PTQ）的二次幂方法，与我们在早期研究 [2] 中提出的方法类似。

2 数据集和模型

NanoGPT [11] 是一个简单、快速的仓库，用于训练和微调生成式预训练 Transformer GPT。其设计遵循 GPT-2 模型，并可在 OpenWebText 数据集上复现 GPT-2。此外，它支持多种设置，便于快速试验和使用较小的数据集，包括莎士比亚数据集。为了获得初步结果，我们选用莎士比亚数据集，并按 80%-10%-10 % 划分为训练集、验证集和测试集，主要关注基于字符的模型，其模型和参数如图 1 所示。这有助于缩短测试和微调的周期，并将重点放在算法以及 PoT 量化感知训练的正确实现上。为了获得更贴近实际的结果，我们将 PoT-QAT 的实验扩展到 GPT-2 模型。基线模型是一个拥有 1.24 亿参数的预训练模型[12]。量化感知训练使用与 GPT-2 训练相同的数据集 OpenWebText 数据集 [13]进行微调。

3 量化感知训练

DNN 的量化有两种不同的方法：训练后量化（PTQ）和量化感知训练（QAT）。在训练后量化中，训练完全以浮点高精度进行，随后在推理期间将权重和激活值量化为较低精度，量化后不再进行进一步调优。训练后量化的复杂度较低，但会遭受更大的性能损失。



Parameter	Character-level	GPT-T2
Block Size (Context size)	64	1024
Number of Layers	4	12
Number of heads	4	12
Embedding Dimension	128	768
Dropout	0.2	0.2
Batch size	12	12
Vocab size	65	50257
Headen Layer dimension	4x128	4x768

Model Parameters

图 1：大语言模型示意图及参数

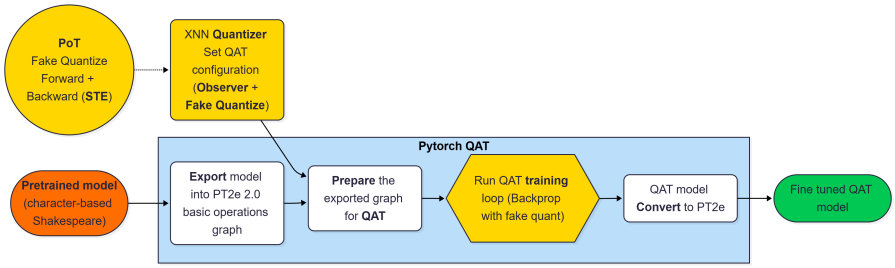


图 2：PyTorch 量化感知训练流程

另一方面，在量化感知训练中，量化是训练过程的一部分：在不同运算之间插入伪量化模块（这些模块通过先量化数值，再进行反量化／重新缩放到浮点范围，来模拟量化损失），使训练能够在考虑量化损失的情况下微调参数。需要注意的是，其中一些运算（例如舍入）不可微分，这给计算反向传播的梯度带来了挑战。一种常用技巧称为直通估计器（STE）：它仅在反向传播路径中将舍入步骤视为恒等映射，从而使梯度能够通过。实践证明，STE 近似效果良好，并使模型能够在训练过程中收敛并适应量化噪声 [14]。

与训练后量化相比，量化感知训练的计算成本更高，但通过额外训练来恢复量化造成的性能下降，效果非常显著。此外，当使用 INT4 等较低位宽，或模型对量化噪声较为敏感时（例如卷积神经网络或 Transformer），量化感知训练尤为重要。由于我们在此研究的是一种非常激进的量化技术（二次幂），因此我们认为，量化感知训练将是改善训练后量化所导致的显著性能下降的一种非常有效的方法 [2]。

我们采用最新的 PyTorch 2 导出模式量化框架 [15-16]，因为它更加灵活且易于扩展。如图 2 所示，PyTorch 中的量化感知训练流程首先将模型导出为由基本运算组成的计算图。接下来，在准备阶段，根据通过量化器模块提供的配置，在每两个运算之间插入伪量化器模块。在量化器模块内部，我们可以配置伪量化运算、量化类型和位宽，以及信号统计信息。具体而言，所谓的观察器模块用于表示信号统计信息（例如最小值和最大值），随后伪量化器可以利用这些统计信息对信号进行缩放及量化/反量化。准备阶段完成后，模型即可开始训练，以在量化影响下调整权重。最后，在训练完成后，通过转换过程移除观察器，仅保留用于推理的伪量化模块。

4 二次幂量化

二次幂量化是一种激进的量化方法，它将量化后的整数限制为二次幂。乘法（逐元素运算、矩阵乘法或卷积中不可或缺的部分）是任何 DNN 处理复杂度的重要组成部分。当其中一个操作数的数值被限制为二次幂时，乘法运算就会转变为简单的位移，从而显著降低计算复杂度。表 1 展示了二次幂量化与均匀仿射量化的公式。二次幂量化的缺点是预计会造成性能损失，这也是引入量化感知训练（QAT）以尝试弥补这一损失的主要动机。

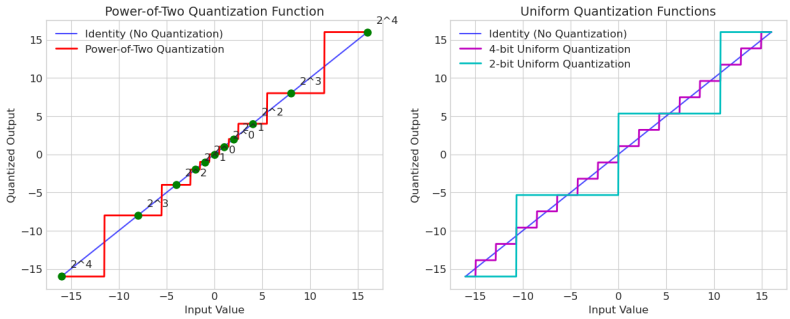


图 3：二的幂（PoT）量化与均匀量化的比较

表 1：量化类型

量化类型	转换公式（输入 x，输出 y）	应用场景
非对称（仿射）	$y = round(x/scale) + zp$	激活值
二次幂（PoT）	$y = 2^{clip(round(\log_2(x/scale)))}$	二的幂权重

与均匀量化一样，二次幂量化的运算不可微，因此我们也需要在反向传播路径中将二次幂量化近似为可微运算。我们研究了直通估计器方法，它似乎是非常好的近似方法；根据初步实验，大语言模型能够收敛。在图 3 中，我们展示了二次幂量化和均匀量化的量化输出。二次幂量化是一种对数方法，其量化步长并不均匀（这也带来了一项额外优势：对较大的离群值可能更有利）。直通估计器假定反向传播路径中的运算为恒等函数，即输出与输入相同（图中的蓝色直线）。

5 实验与性能评估

5.1 使用莎士比亚数据集的字符级模型

在初步试验中，我们考察了字符级大语言模型的量化感知训练（QAT）性能，其中权重分别采用均匀量化和二次幂（PoT）量化（请注意，要发挥 PoT 的优势，只需将权重量化为二次幂）。表 2 展示了训练过程中训练/验证损失的收敛情况。我们先使用浮点数训练模型 15000 次迭代，$LR = 10^{-3}$ 使交叉熵损失从略高于 4 降至 1.6。随后，我们将训练好的模型载入量化感知训练流程，并分别采用均匀量化和 PoT 量化进行训练（$LR = .5 * 10^{-5}$）。表中展示了均匀量化（4 比特 [-8，7], ）与 9 级 PoT 量化的结果。结果表明，两种方法都能明确收敛，证明 PoT 可通过量化感知训练得到改进。具体而言，应用量化后，性能会先有所下降（损失从 1.6 的基线升至 1.8 以上），随后经过若干次训练迭代又重新收敛。

5.2 使用 OpenWebText 数据集的 GPT-2 124M 模型

我们扩展了 PoT-QAT 在 GPT-2 124M 模型上的结果。我们使用不同的指标分析性能，包括训练/验证损失收敛曲线、困惑度和 BERT-Score。

5.2.1 训练/验证损失收敛

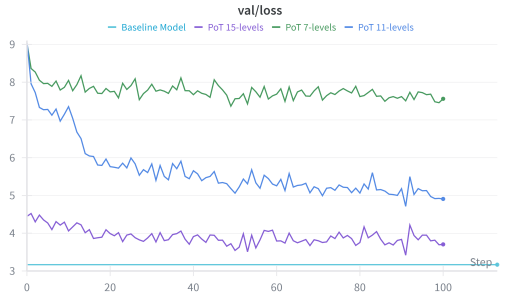
如图 4a 和 4b 所示，训练损失和验证损失均成功收敛。我们尝试了不同的学习率 LR，最终选定了 $.5e-5$。上下文长度为 64，批量大小设为 12（我们测试了 12、24、48 和 64 等不同批量大小）。我们尝试了不同分辨率的二次幂量化（7、11 和 15 个级别）。采用 15 个级别的二次幂量化（$[-2^7, 2^7]$）能够

表 2：nano-GPT 字符级大语言模型量化感知训练性能。均匀量化与二次幂的比较

迭代	**均匀量化**		**二次幂**	
	训练	验证	训练	验证
15500	1.61635	**1.84743**	1.65916	**1.82383**
19500	1.56602	1.68807	1.63777	1.75756
22500	1.61594	**1.59317**	1.66232	**1.68218**



(a)



(b)

图4：不同二次幂设置下量化感知训练的收敛情况。每一步包含5个训练步迭代。(a) 训练损失随迭代次数的变化。(b) 验证损失随迭代次数的变化。

表3：二次幂训练后量化（PTQ）与二次幂量化感知训练（QAT）的困惑度表现

量化范围		二次幂 - 训练后量化		二次幂 - 量化感知训练	
整数范围	N 比特	l_{ce}	困惑度	l_{ce}	困惑度
基线 float32	32 比特	3.167	23.73	3.167	23.167
$[-2^3, 2^3]$	3 比特	9.01	8184	7.42	1669
$[-2^5, 2^5]$	4 比特	8.76	6374	4.71	111
$[-2^7, 2^7]$	**4 比特**	4.5	**90**	3.417	**30.47**

缩小相较于基线模型的性能差距。与其他二次幂量化变体相比，15 个级别的二次幂量化初期偏差较小，但在训练后期观察到轻微过拟合。另一方面，7 个和 11 个级别的二次幂量化偏差相对较大，且随着训练而加剧，尤其是 11 个级别的情况（继续训练 11 个级别或许能进一步改善）。

5.2.2 困惑度

困惑度是衡量模型预测下一个词元准确程度的指标。换句话说，它衡量模型学习数据分布的能力。顾名思义，困惑度越低，表示模型能够以更少的不确定性进行预测。它计算为交叉熵损失 l_{ce}的指数，即困惑度为 exp(l_{ce})

为了衡量交叉熵，我们对从验证集捕获的多个批次进行多次迭代并取平均值。输入为 k–维逻辑值 $t \in \mathbb{R}^k$、目标类别为 y 时的交叉熵 $l_{ce}(t, y)$ 定义如下：

$$l_{ce}(t, y) = -\log P(y; t) = -\log\left(\frac{\exp(t_y)}{\sum_s \exp(t_s)}\right)$$

在表3中，我们展示了训练后量化（PTQ）所得到的二次幂（PoT）量化模型的困惑度（即未经额外训练的二次幂量化，属于先前结果），并与本文提出的量化感知训练（QAT）进行对比。结果表明，经过训练（QAT）后，模型性能显著提升。例如，对于 15 级二次幂量化（相当于 4 比特），QAT 能将困惑度从 90（PTQ）降至 30，提升幅度显著（超过 66%），且非常接近基线模型的困惑度 23.7。

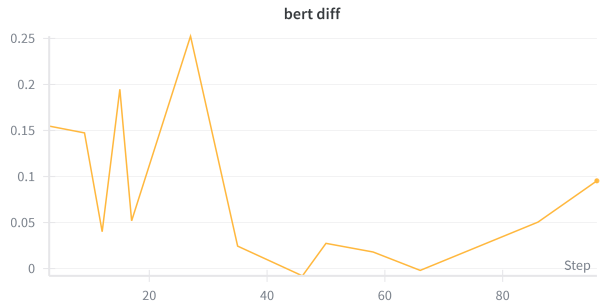


图5：在验证损失为 ≈ 3.78时，PoT-QAT量化结果与基线浮点结果的BERT-Score差值

5.2.3 BERT-Score

BERT-Score 衡量候选序列中的每个词元与参考序列中的每个词元之间的相似度分数，其中词元相似度是通过上下文嵌入而非精确匹配来计算的 [17]。BERT-Score 是人工判断的良好指标。我们使用 Hugging Face evaluate软件包，以基础模型“roberta-base”计算 BERT-Score。我们对 10 次迭代取平均值；每次迭代都会生成一段新文本并计算 BERT-Score。在图5中，我们绘制了基线浮点模型与经过训练的量化模型之间的 BERT-Score 差值随训练迭代的变化情况，只报告验证表现有所改善的检查点结果。基线模型与参考数据之间的 BERT-Score，以及量化模型与参考数据之间的 BERT-Score，均分别计算。基线模型的平均 BERT-Score 约为 0.2，虽然不算很高，但考虑到 GPT-2 124M，这一结果并不意外。从曲线可以清楚看出，随着训练推进，量化后的 GPT-2 与基线 GPT-2 之间的差距逐渐缩小，直至在验证迭代 66（训练步 330）左右达到最小的 BERT-Score 差值（0.0019 的差异 ≈ 1 % 损失）。值得注意的是，最佳 BERT-Score 并未出现在验证误差最低时（最佳 BERT-Score 对应的验证误差为 ≈ 3.78，而最低验证误差为 3.41），这可能表明训练循环后期出现了一定程度的过拟合。这对于选出最佳模型非常有用，因为应综合考虑所有指标。作为参考，我们在附录的图7a和图7b中展示了基础模型和量化模型生成文本的示例。

6 内存与计算节省

表4总结了使用4位PoT量化所节省的内存和计算量。内存节省源于使用较小的位宽（PoT中的4比特与基线 float32中的32比特相比），这与任何4位量化技术类似。不过，计算节省既来自在所有矩阵乘法中使用较小位宽的权重（速度提升2至5倍），也来自用位移操作取代乘法（额外提升1.5至2倍速度）。

表4：GPT-2 124M：FP32与4位二的幂量化

指标	FP32	4 位二次幂
模型大小	124 MB x 4 = 496 MB	62 MB（节省 87.5%）
推理速度	1x	快 3–10 倍

7 结论与未来工作

我们提出了针对大语言模型权重的二次幂量化，可显著降低内存占用和计算复杂度。我们提出使用量化感知训练（QAT），以便通过额外训练缓解激进量化造成的性能下降。我们在用于 nano-GPT 代码的 PyTorch QAT 框架中实现了二次幂（PoT）功能。GPT-2 124M 的结果表明，与未经训练的量化（训练后量化，PTQ）相比，采用 QAT 的 PoT 效果显著提升。15 级 PoT 的困惑度较 PTQ 提升了 66%。BERT-Score 也显示，经过训练循环后性能明显提升，且与基线模型相比，BERT-Score 的差距接近 1%。未来的工作包括在 Llama 和通义千问等更大的模型上测试 PoT-QAT。此外，在设备上运行该算法并测量实际的复杂度节省也会非常有意义。

贡献

• 为 nano-GPT 大语言模型实现 QAT 框架，并配置/优化模型参数（如学习率）和量化参数，以实现收敛。
• 实现二次幂可微类，其中包含前向和反向函数。
• 将二次幂类集成到量化感知训练框架中（继承伪量化类）。
• 作为第一步，在字符级大语言模型上实现均匀量化和二次幂量化感知训练的收敛。

• 将结果扩展到更贴近实际的 GPT-2 124M 1.24亿参数模型。对于不同分辨率的多个二次幂版本，实现了 PoT-QAT 的收敛。
• 使用训练/验证损失、困惑度和 BERT-Score 等多种指标评估性能。

• 最后，性能指标表明，量化感知训练在帮助二次幂量化缩小与基线模型之间的性能退化差距方面取得了巨大成功（与训练后量化相比，困惑度提升了 66%；与全精度相比，BERT-Score 仅下降 1 %）

参考文献

[1] 什么是量化大语言模型？"https://www.tensorops.ai/post/what-are-quantized-llms#viewer-2tmd3"

[2] Elgenedy, Mahmoud.《大语言模型中的二次幂（PoT）权重》。arXiv 预印本 arXiv:2506.00315（2025）。

[3] Liu, Zechun、Barlas Oguz、Changsheng Zhao、Ernie Chang、Pierre Stock、Yashar Mehdad、Yangyang Shi、Raghuraman Krishnamoorthi 和 Vikas Chandra。《大语言模型量化感知训练：无需数据的量化感知训练》。载于《计算语言学协会：ACL 2024 论文集》，第 467–484 页。2024 年。

[4] Gholami, Amir, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney 和 Kurt Keutzer。《高效神经网络推理的量化方法综述》。收录于《低功耗计算机视觉》，第 291–326 页。Chapman and Hall/CRC，2022 年。

[5] Krishnamoorthi, Raghuraman。《面向高效推理的深度卷积网络量化：白皮书》。arXiv 预印本 arXiv:1806.08342（2018 年）。

[6] Wu, Hao, Patrick Judd, Xiaojie Zhang, Mikhail Isaev 和 Paulius Micikevicius。《深度学习推理的整数定量化：原理与实证评估》。arXiv 预印本 arXiv:2004.09602（2020 年）。

[7] Elhoushi, Mostafa, Zihao Chen, Farhan Shafiq, Ye Henry Tian 和 Joey Yiwei Li。《Deepshift：迈向无乘法神经网络》。发表于 IEEE/CVF计算机视觉与模式识别会议论文集，第 2359–2368 页。2021 年。

[8] Przewlocka-Rus, Dominika、Syed Shakib Sarwar、H. Ekin Sumbul、Yuecheng Li 和 Barbara De Salvo。《面向低位宽和硬件兼容神经网络的二次幂量化》。arXiv 预印本 arXiv:2203.05025（2022）。

[9] McDanel, Bradley、Sai Qian Zhang、H. T. Kung 和 Xin Dong。《使用二次幂权重加速卷积神经网络的全栈优化及现场可编程门阵列验证》。载于 ACM国际超级计算会议论文集，第 449–460 页。2019。

[10]Wang, Xinyu、Vahid Partovi Nia、Peng Lu、Jerry Huang、Xiao-Wen Chang、Boxing Chen 和 Yufei Cui。《PoTPTQ：面向大语言模型的两步二次幂训练后量化》。arXiv 预印本 arXiv:2507.11959（2025）。

[11]https://github.com/karpathy/nanoGPT，作者：Karpathy

[12] https://huggingface.co/openai-community/gpt2

[13] https://huggingface.co/datasets/Skylion007/openwebtext

[14] https://api.wandb.ai/links/byyoung3/eb1eylcu

[15] https://pytorch.org/docs/stable/quantization.html#quantization-api-summary

[16] （原型版）PyTorch 2.0 导出后训练静态量化 — PyTorch教程 2.0.1+cu117 文档

[17]Zhang, Tianyi、Varsha Kishore、Felix Wu、Kilian Q. Weinberger 和 Yoav Artzi。《BERTScore：使用 BERT 评估文本生成》。arXiv 预印本 arXiv:1904.09675 （ 2019 年）。

8 致谢

所有工作均由我本人 Mahmoud Elgenedy 独立完成，没有其他团队成员。谨此感谢 CS230 的授课教师和助教给予的帮助与支持。LaTeX 模板在以下模板的基础上略作修改：“CS230：深度学习，2018 年冬季，斯坦福大学，加利福尼亚州。（LaTeX 模板借自 NIPS 2017。）”我通过互联网搜索和使用大语言模型，获得了一些基础问题方面的帮助。

附录

8.1 代码

所有代码均已上传至以下 GitHub 仓库：
https://github.com/melgenedy/Quantized_nano_GPT-QAT-POT.git

8.2 量化模型的部分代码片段

opcode	name	target	args	kw
get_attr	transformer_wpe_weight	transformer.wpe.weight	()	{}
get_attr	transformer_h_0_ln_1_weight	transformer.h.0.ln_1.weight	()	{}
get_attr	_frozen_param0	_frozen_param0	()	{}
call_function	dequantize_per_tensor_default	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param0, 0.06372686475515366, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param1	_frozen_param1	()	{}
call_function	dequantize_per_tensor_default_1	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param1, 0.016682995483279228, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_0_ln_2_weight	transformer.h.0.ln_2.weight	()	{}
get_attr	_frozen_param2	_frozen_param2	()	{}
call_function	dequantize_per_tensor_default_2	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param2, 0.043351493775844574, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param3	_frozen_param3	()	{}
call_function	dequantize_per_tensor_default_3	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param3, 0.04125349968671799, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_1_ln_1_weight	transformer.h.1.ln_1.weight	()	{}
get_attr	_frozen_param4	_frozen_param4	()	{}
call_function	dequantize_per_tensor_default_4	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param4, 0.03207191079854965, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param5	_frozen_param5	()	{}
call_function	dequantize_per_tensor_default_5	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param5, 0.021004071459174156, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_1_ln_2_weight	transformer.h.1.ln_2.weight	()	{}
get_attr	_frozen_param6	_frozen_param6	()	{}
call_function	dequantize_per_tensor_default_6	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param6, 0.036302134394645694, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param7	_frozen_param7	()	{}
call_function	dequantize_per_tensor_default_7	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param7, 0.04405483976006508, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_2_ln_1_weight	transformer.h.2.ln_1.weight	()	{}
get_attr	_frozen_param8	_frozen_param8	()	{}
call_function	dequantize_per_tensor_default_8	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param8, 0.0368971973657608, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param9	_frozen_param9	()	{}
call_function	dequantize_per_tensor_default_9	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param9, 0.0228872112929821, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_2_ln_2_weight	transformer.h.2.ln_2.weight	()	{}
get_attr	_frozen_param10	_frozen_param10	()	{}
call_function	dequantize_per_tensor_default_10	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param10, 0.03949552774429321, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param11	_frozen_param11	()	{}
call_function	dequantize_per_tensor_default_11	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param11, 0.048228416591882706, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_3_ln_1_weight	transformer.h.3.ln_1.weight	()	{}
get_attr	_frozen_param12	_frozen_param12	()	{}
call_function	dequantize_per_tensor_default_12	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param12, 0.036666844044015541, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param13	_frozen_param13	()	{}
call_function	dequantize_per_tensor_default_13	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param13, 0.030309339985251427, 0, -8, 7, torch.int8)	{}
get_attr	transformer_h_3_ln_2_weight	transformer.h.3.ln_2.weight	()	{}
get_attr	_frozen_param14	_frozen_param14	()	{}
call_function	dequantize_per_tensor_default_14	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param14, 0.0433604009449481960, 0, -8, 7, torch.int8)	{}
get_attr	_frozen_param15	_frozen_param15	()	{}
call_function	dequantize_per_tensor_default_15	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param15, 0.09454374015331268, 0, -8, 7, torch.int8)	{}
get_attr	transformer_ln_f_weight	transformer.ln_f.weight	()	{}
get_attr	lm_head_weight	lm_head.weight	()	{}
get_attr	_frozen_param16	_frozen_param16	()	{}
call_function	dequantize_per_tensor_default_16	quantized_decomposed.dequantize_per_tensor.default	(_frozen_param16, 0.08917881548404694, 0, -8, 7, torch.int8)	{}
get_attr	lifted_tensor_0	lifted_tensor_0	()	{}
placeholder	idx	idx	()	{}
call_function	arange	aten.arange.start	(0, 64)	{'
call_function	embedding	aten.embedding.default	(lm_head_weight, idx)	{}
call_function	quantize_per_tensor_default_17	quantized_decomposed.quantize_per_tensor.default	(embedding, 0.0026914246845990042, -1, -128, 127, torch.int8)	{}
call_function	dequantize_per_tensor_default_17	quantized_decomposed.dequantize_per_tensor.default	(quantize_per_tensor_default_17, 0.0026914246845990042, -1, -128, 127, torch.int8)	{}
call_function	embedding_1	aten.embedding.default	(transformer_wpe_weight, arange)	{}
call_function	quantize_per_tensor_default_18	quantized_decomposed.quantize_per_tensor.default	(embedding_1, 0.004223781637847423 6, 68, -128, 127, torch.int8)	{}
call_function	dequantize_per_tensor_default_18	quantized_decomposed.dequantize_per_tensor.default	(quantize_per_tensor_default_18, 0.004223781637847423 6, 68, -128, 127, torch.int8)	{}
call_function	add	aten.add.Tensor	(dequantize_per_tensor_default_17, dequantize_per_tensor_default_18)	{}
call_function	quantize_per_tensor_default_19	quantized_decomposed.quantize_per_tensor.default	(add, 0.005077532492578 03, 43, -128, 127, torch.int8)	{}
call_function	dequantize_per_tensor_default_19	quantized_decomposed.dequantize_per_tensor.default	(quantize_per_tensor_default_19, 0.005077532492578 03, 43, -128, 127, torch.int8)	{}
call_function	dropout_558	aten.dropout.default	(dequantize_per_tensor_default_19, 0.5, True)	{}
call_function	quantize_per_tensor_default_20	quantized_decomposed.quantize_per_tensor.default	(dropout_558, 0.008676772005856037, 45, -128, 127, torch.int8)	{}
call_function	dequantize_per_tensor_default_20	quantized_decomposed.dequantize_per_tensor.default	(quantize_per_tensor_default_20, 0.008676772005856037, 45, -128, 127, torch.int8)	{}
call_function	layer_norm	aten.layer_norm.default	(dropout_558, [128], transformer_h_0_ln_1_weight)	{}
call_function	quantize_per_tensor_default_21	quantized_decomposed.quantize_per_tensor.default	(layer_norm, 0.039218727377395 63, 18, -128, 127, torch.int8)	{}

图 6：以表格形式打印的量化模型片段

8.3 量化模型与基线 GPT-2 模型生成文本的部分片段

```
print(decode(y_org[0].tolist()),end="")
. There are formulations of quantum mechanics where collapse of the wave function is truly random and unpredictable; there are others with
hidden variables, in which the true state of the universe isn't defined by the wave function. In any of those cases, our analysis is comple
tely beside the point. It's interesting to note that the best single-variable analysis of the state of the universe is by applying a discre
te discrete state property. And that is what makes it in our view relevant to our discussion. But what is important is to understand why th
at is so.

Why? Well the evidence suggests that because space is finite, there
```

(a)

```
print(decode(y_last[0,:-1].tolist()),end="")

. There are formulations of quantum mechanics where collapse of the wave function is truly random and unpredictable; there are others with
hidden variables, in which the true state of the universe isn't defined by the wave function. In any of those cases, our analysis is comple
tely beside the point. It's interesting to note that quantum mechanics will be able to explain so much of the cosmological system in such a
way's natural system's laws of space, in particular, which to illustrate these phenomena in sequence's complexity, as reflected in these gr
avitational forces. In light of these equations, these theories
```

(b)

图 7：上下文长度为 64（已突出显示）时生成的文本，温度为 $= 0.8$。量化模型在验证损失为 3.76 时生成的文本，与基线模型相比看起来非常合理。请注意，基线模型是 GPT-2 124M，相较于更新或参数量更大的模型而言，它相对较小，因此即使是基础模型生成的文本也有所局限，但对于概念验证而言非常有用。(a) 基线 GPT-2 1.24亿参数，浮点。(b) 验证损失为 ≈ 3.7 时的二次幂量化感知训练量化模型。