LLM.265: Video Codecs are Secretly Tensor Codecs

Ceyu Xu*
Duke University
Durham, North Carolina, USA
ceyu.xu@duke.edu

Yongji Wu*
Duke University
Durham, North Carolina, USA
yongji.wu769@duke.edu

Xinyu Yang*
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
xinyuya2@andrew.cmu.edu

Beidi Chen
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
beidic@andrew.cmu.edu

Matthew Lentz
Duke University
Durham, North Carolina, USA
mlentz@cs.duke.edu

Danyang Zhuo
Duke University
Durham, North Carolina, USA
danyang@cs.duke.edu

Lisa Wu Wills
Duke University
Durham, North Carolina, USA
lisa@cs.duke.edu

Abstract

As the parameter size of large language models (LLMs) continues to expand, the need for a large memory footprint and high communication bandwidth have become significant bottlenecks for the training and inference of LLMs. To mitigate these bottlenecks, various tensor compression techniques have been proposed to reduce the data size, thereby alleviating memory requirements and communication pressure.

Our research found that video codecs, despite being originally designed for compressing videos, show excellent efficiency when compressing various types of tensors. We demonstrate that video codecs can be versatile and general-purpose tensor codecs while achieving the state-of-the-art compression efficiency in various tasks. We further make use of the hardware video encoding and decoding module available on GPUs to create a framework capable of both inference and training with video codecs repurposed as tensor codecs. Building on insights gained from video codecs, we further show that the hardware of the video codecs can be customized and enhanced to significantly improve tensor encoding/decoding throughput without incurring substantial costs, making it a highly effective solution for large-scale model deployment without requiring significant modifications to the existing GPU architecture.

CCS Concepts

• **Computer systems organization → Architectures**; • **Computing methodologies → Artificial intelligence**; **Image compression**; • **Mathematics of computing → Coding theory**; • **Hardware**;

*Ceyu Xu, Yongji Wu, and Xinyu Yang are co-first authors of this work.

Keywords

Large Language Models, Video Codecs, Model Compression, Distributed Training

ACM Reference Format:
Ceyu Xu, Yongji Wu, Xinyu Yang, Beidi Chen, Matthew Lentz, Danyang Zhuo, and Lisa Wu Wills. 2025. LLM.265: Video Codecs are Secretly Tensor Codecs. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3725843.3756078

1 Introduction

Recently, Large language models (LLMs) have achieved significant success, showcasing remarkable proficiency in various applications, such as virtual assistants [60], chatbots [60], and automated customer service platforms [63].

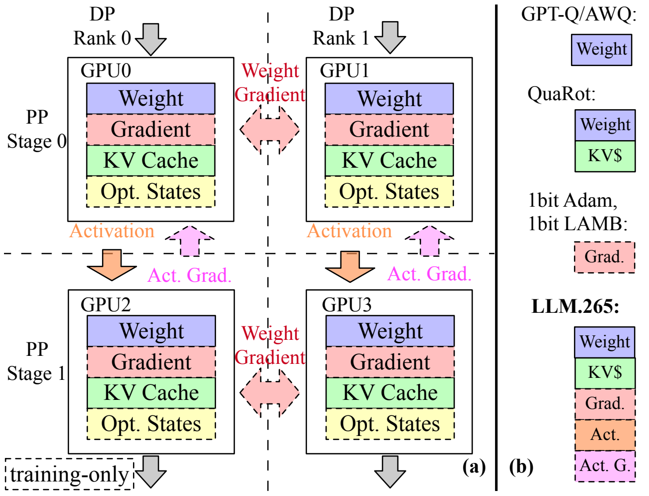

Figure 1: LLM.265: General-Purpose and Versatile Tensor Compression for LLM Training and Inference.

As their parameter size grows, LLMs develop emergent abilities [83]. These abilities enable them to carry out more advanced and complex tasks such as code generation [49, 68], mathematical problem-solving [70], and theorem proving [65], even some they weren't explicitly trained for. This growing range of applications has motivated researchers to train increasingly larger models, such as GPT4 [60], Nemotron-4-340B [32], and LLaMA-3-70B [1], which further leads to a revolution in the development and deployment of AI systems and hardware.

The training and inference of these large language models with large parameter sizes often strain the underlying computing infrastructure, posing challenges in terms of memory capacity and communication bandwidth. For example, inferencing a DeepSeek-V3-671B [17] model requires at least 671GB memory for storing the parameters only, far exceeding the capacity of a single GPU. To address this problem, parallelism strategies such as pipeline parallelism (PP) [30, 31] and data parallelism (DP) [11, 16] have been developed to distribute models across multiple GPUs and increase throughput for inference and training. Figure 1(a) shows an example of a distributed LLM using pipeline parallelism and data parallelism. However, inter-GPU communication is required for such distributed models. As shown in Figure 1(a), activations must be transmitted between pipeline stages for distributed inference. During training, communication pressure is even higher because in addition to activations, both weight gradients and activation gradients must be transmitted across GPUs.

While scaling up systems and hardware for LLM workloads remains challenging, optimizing data movement and storage becomes an overarching goal, of which compression has become a crucial strategy. Compression trades more computation for a reduced amount of data to be maintained, which subsequently translates to a reduced memory footprint and reduced pressure on the communication system. Figure 1(b) lists the tensors required for inference and training of LLMs. Traditionally, these tensors are stored, inferred, and trained in half-precision float (FP16) or Brain-float (BF16) [37] formats, with each value occupying two bytes. Research has shown that with compression, model weights can be reduced to 3-4 bits [25, 42] and gradients to 3 bits [41, 77] without significant accuracy degradation. However, existing tensor compression techniques also face specific challenges: they are not *general-purpose* [25, 41, 42, 77] and lack *versatility*. As illustrated in Figure 1(b), a variety of tensors, such as model weights, gradients, and activations, need to be maintained. None of the existing approaches is general-purpose, typically requiring different compression algorithms for each tensor type. In addition, some approaches [25, 41, 42, 77, 85] are not versatile due to their reliance on data-aware calibrations and warm-up periods. This dependence complicates deployment and limits robustness when calibration data is unknown or biased. Moreover, existing algorithms typically compress values into short integers, restricting each compressed value to using an integer number of bits, which makes achieving a fractional bitrate (defined as the average number of bits in the compressed form per value in the uncompressed form) impossible, further limiting their versatility.

Our insight is that video codecs (consisting of both an encoder and a decoder), despite being designed for video compression, work well and even achieve state-of-the-art information efficiency for various tensor compression tasks. Specifically, we found the distribution of the values representing pixels in videos shares characteristics with tensors in LLMs, allowing video codecs to compress tensors efficiently with only minimal adjustment of codec parameters. Modern GPUs are equipped with on-chip video encoding and decoding engines (e.g., NVENC/NVDEC on Nvidia GPUs [59]), which are typically idle during LLM workloads. This provides an opportunity to directly leverage these unused resources for optimal tensor compression throughput, essentially allowing us to utilize them at no additional cost. We refer to our method of using video codecs for tensor compression as **LLM.265**. LLM.265 is general-purpose: it offers a unified compression method that effectively compresses various types of tensors while achieving state-of-the-art information efficiency across all tasks. LLM.265 is also versatile: being data-independent, LLM.265 requires no data-aware calibration or warm-up. In addition, LLM.265 works at fractional bitrates (e.g. 2.3 bits per value), not limited to integer bitrates. Thanks to these properties, LLM.265 allows for extreme LLM compression by simultaneously compressing multiple types of tensors, while providing the capability of fine-grained fractional bitrate tuning for ultimate information efficiency. The reduced memory footprint and communication bandwidth requirement enable the training and inference of large models on commodity-level GPUs with limited resources. We demonstrate that LLM.265 is the first method capable of conducting inference for the LLaMa-3-70B [1] model with a sequence length of 128k on 4 × 8GB devices, using 3.5 bits per value for communication and 2.9 bits per value for weight and key-value (KV) cache compression. In contrast, previous methods only compress weight and KV cache to 4 bits at most and do not consider activation compression for communication.

In this paper, we will first provide empirical evidence demonstrating the effectiveness of video codecs in tensor compression. Our experiments then showcase how LLM.265 can compress the weights, KV cache, and activations of LLMs, thereby reducing the memory footprint and communication cost during inference. To highlight the general-purpose capability of LLM.265, we further show its efficacy in compressing gradients to reduce communication size during distributed training under pipeline and data parallelism. Finally, we will discuss the insights LLM.265 offers for future GPU and accelerator hardware design, exploring the potential for developing custom hardware codecs specifically tailored for tensors. We make the following contributions:

(1) We demonstrate that video codecs such as H.264 and H.265 are highly effective for compressing various types of tensors, including weights, activations, and gradients.
(2) We present empirical evidence demonstrating the effectiveness of video codecs for tensor compression, providing valuable insights to guide the development of future tensor compression algorithms.
(3) We develop LLM.265 that leverages the hardware video codecs in modern GPUs to compress tensors during LLM inference and training. It achieves compression ratios of up to 3-20× while delivering superior throughput compared to state-of-the-art compression methods.

(4) We further show that video codecs can be augmented into more efficient *tensor codecs* specialized for tensor compression with only a marginal increase in cost, reducing die area and power consumption. We propose the integration of tensor-specialized codecs into future GPU and accelerator designs.

Our implementation is publicly available at https://github.com/Entropy-xcy/llm.265.

2 Background

2.1 Model Compression

Quantization techniques have been widely applied in model compression [7, 29, 36, 55]. Most current quantization methods are based on vanilla round-to-nearest quantization (RTN). Given a tensor T, the RTN quantization function is defined as: $Q(\mathbf{w}) = \Delta \cdot \text{Round}\left(\frac{\mathbf{w}}{\Delta}\right)$, $\Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$, where N is the number of quantization bits, and Δ is the quantization scaler determined by the absolute maximum value. Additionally, recent work has also explored non-uniform quantization techniques such as K-means clustering [39], vector quantization [80], and NormalFloat quantization [20]. While our work primarily focuses on dense compression methods, several other studies have explored sparse model compression for both training and inference [72, 75, 89]. These methods are orthogonal to our methods. We leave the discussion and comparison of these work for future research.

Weight Compression: In LLM inference, the size of model weights can be a bottleneck. Modern LLMs can scale up to over 600 billion parameters [17], requiring a total of more than one terabyte of GPU memory to store both the model weights and the KV-cache. This would involve distributing the model across 16 to 32 Nvidia GPUs [17], each with 80GB of memory, which would incur significant costs and create considerable communication overhead. To address this issue, LLMs are usually compressed with quantization-aware training (QAT) methods [20, 47, 86] or post-training quantization (PTQ) techniques [10, 25, 42, 80] to reduce their memory footprints such that they could be served with less number of GPUs. QAT approaches require additional training, while PTQ algorithms calibrate the rounding of weights into low-bit integers by running the model through a calibration dataset. Through this calibration process, weights can be compressed from 16 bits to 3-4 bits, while still maintaining acceptable model accuracy.

Activation Compression: Simply compressing the model weights is not sufficient. During inference, especially in scenarios involving long-context lengths and large batch sizes, the size of the activations (including KV cache) also becomes a bottleneck. Dual-side quantization and compression techniques, such as SmoothQuant [85] and QServe [44], have been developed to compress both model weights and activations. Activation compression presents a greater challenge due to the significant outliers that exist in the activation's distribution [85]. As a result, current activation compression algorithms typically achieve 8-bit compression without accuracy loss [44]. Some methods reach 4-bit compression [4, 48], but often with degraded accuracy. These approaches primarily focus on compressing the activations before matrix multiplication in linear layers, thereby accelerating the GEMM Kernel on modern GPUs. However, none of them explore compressing activations transmitted between machines to reduce the communication cost, which is much more time-consuming in distributed settings. In our work, we consider compressing the KV cache during inference, as well as the activations between different pipeline parallelism stages in both training and inference.

Gradient Compression: Model compression in training becomes more complex during backward propagation stages, where maintaining gradients is a major bottleneck for memory and communication. In distributed training, gradients need to be exchanged across machines, often causing communication bottlenecks. 1-bit Adam [77] and 1-bit LAMB [41] compress the weight gradient to an average of 3-4 bits using a two-stage approach. In the warm-up stage, 16-bit floating point values are transmitted without compression, as the model hasn't converged to a point where the weights can be easily compressed yet. Once the model convergence becomes stable, these algorithms enter a variance-freeze stage, where they can compress the weight gradients to 1 bit per value. Notably, these methods only support weight gradient compression while not supporting activation gradient compression, limiting their use case to data parallelism but not pipeline parallelism as pipeline parallelism requires communication of activation gradients. In our work, we apply our method to compress both weight gradients in data parallelism and activation gradients in pipeline parallelism.

Problems of Existing Approaches: We have identified two main issues of existing approaches. First, existing tensor compression algorithms are not general-purpose, with each algorithm only capable of compressing one or two types of tensors. This makes it complex to create a "compress-everything" system, and the quality of the results when using these algorithms in conjunction is unverified. Second, all these algorithms require specific data-dependent calibrations and parameter tuning, making the system design more complex and raising concerns about their robustness across different models and varying data distributions.

2.2 AVC and HEVC Video Codecs

Modern encoding techniques such as Advanced Video Coding [34] (AVC or H.264) and High-Efficiency Video Coding [35] (HEVC or H.265) enable efficient video streaming over limited network bandwidth by compressing raw video footages up to a ratio of 1000:1 with unnoticeable quality loss of the videos. A video codec consists of an encoder that compresses raw videos to compressed bitstreams and a decoder that reconstructs every pixel from the bitstreams. The H.264 and H.265 standards establish a set of fixed rules for the decoding process while allowing flexibility in implementing the encoding process. The encoding pipeline is a compounding of several unique compression blocks. Figure 2 (a) shows a typical H.265 encoding pipeline implementation. The encoding process begins with raw video frames. First, a process called the ①CTU (code tree unit) partitioning divides the video into a Quad-Tree of Coding Units (CUs) [73]. Next, the encoder exploits spatial and temporal correlations through ③intra-frame prediction and ④⑤inter-frame prediction within each CU, allowing redundant pixel information to be eliminated when pixels can be accurately predicted from neighboring pixels or reference frames. The residual between the actual pixels and the prediction is measured, transformed, quantized, and

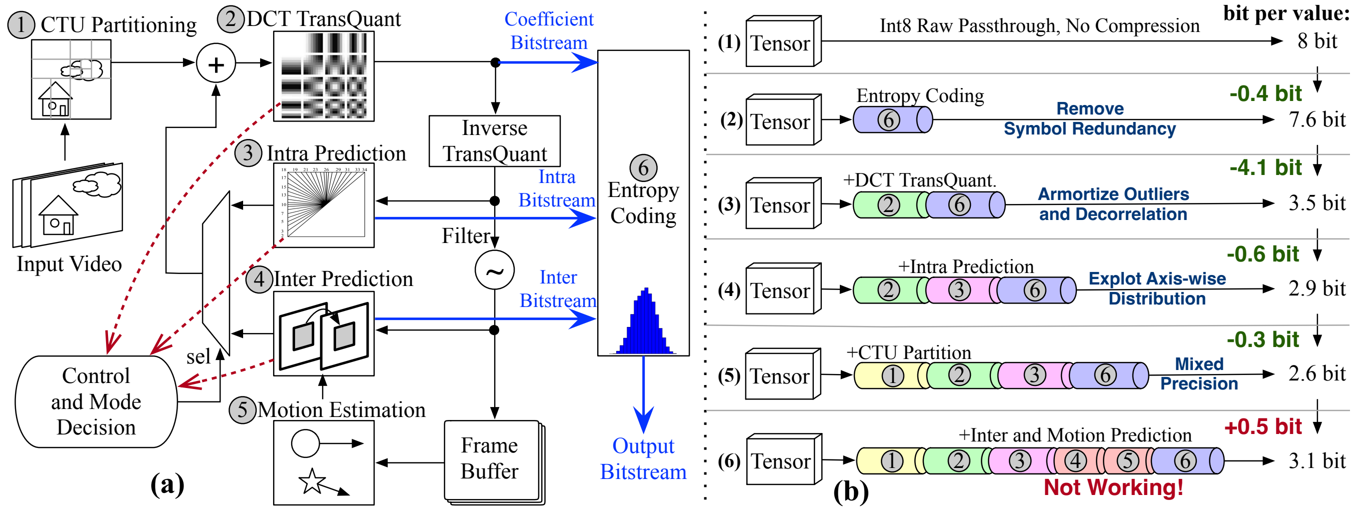

Figure 2: Why does the Video Codec Work for LLM? (a) illustrates the pipeline of the H.265 video encoder. In (b), we incrementally activate the stages in the H.265 video encoding pipeline to demonstrate how each step contributes to the compression process. We constrain the quality of the compression/decompression process to have a maximum mean square error of 0.01.

stored as coefficients. This step is called the ②DCT transform. Finally, all the predictive states, coefficients, and meta-data are input into an ⑥entropy coder (e.g., CABAC [51]) to exploit system-level symbol redundancies.

3 Video Codecs are secretly Tensor Codecs

3.1 Why do Video Codecs Work for Tensor?

Video codecs achieve high-quality and efficient compression by leveraging prediction. The idea is that **the majority of pixels can be predicted, leaving only sparse and small residuals to be encoded.** In addition to the prediction, steps such as Discrete-Cosine Transform (DCT), quantization, and entropy coding exploit other types of redundancies in the video that are either imperceptible to the human eye or can be masked due to the non-uniform distribution of symbols.

Our work demonstrates that some stages in the video coding pipeline are also effective for compressing tensors. To analyze why video codecs work and how each stage in the pipeline contributes to the compression of large language model tensors, we set up an experiment where we enabled the stages in the encoding pipeline step-by-step, as shown in Figure 2 (b) steps (1) to (6). In our experiment, we used the weight tensor of the Key-Projection linear layer in the LLaMA-2-7B [78] network as an example to test if video codec works for compressing tensors. [1] Video codecs like H.264 and H.265 allow users to set the bitrate target explicitly. We constrained the maximum distortion to a mean square error (MSE) of less than 0.01. Detailed analysis of how the bitrate of the codec and the distortion of the weight will affect the LLM's accuracy will be shown in §4. We sweep the bitrate from low to high for each codec pipeline setting until we find a bitrate that achieves this quality

[1]We constructed a 4D video tensor from the 2D weights of the Key-Projection linear layer, using the layer index as the temporal channel and only the Luma channel for gray-scale encoding, with Chroma channels padded with zeros.

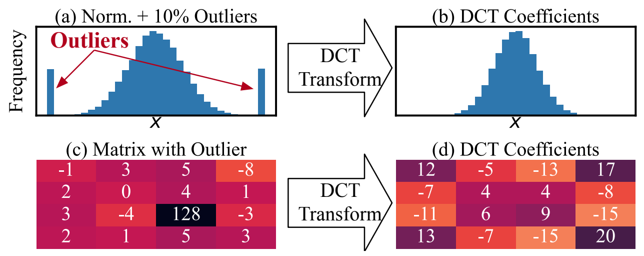

Figure 3: Transform coding mitigates encoding outliers by mapping them to all values within the block. The transition from (a) to (b) demonstrates how DCT removes outliers from a normal distribution containing outliers. (c) to (d) gives an example of an outlier with a value of 128 being mitigated.

constraint. We demonstrated that incrementally activating stages in this pipeline reduced the average bits per value from 8 bits to 2.6 bits for achieving a quality of MSE < 0.01.

Entropy Coding: Entropy coding is a lossless compression technique used in various compression algorithms. It assigns shorter codes to more frequent symbols and longer codes to less frequent symbols. In the context of video codecs, entropy coding can exploit redundancies in the distribution of symbols, reducing data size without introducing additional distortion due to its lossless nature. The effectiveness of entropy coding in compressing videos can be generalized to compressing tensors. As prior works have shown, weights, activations, and gradients in LLM training and inference all conform to a normal or bell-shaped distribution [20, 42, 45]. The non-uniformity in symbol distribution allows entropy coding to achieve an average reduction of 0.4 bits per value for the weight tensor, as illustrated in Figure 2 (b) step (2).

Transform Coding: Transform coding is a vital technique used in video and image codecs. It automatically de-correlates pixels in

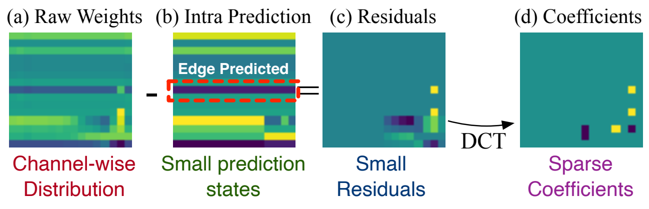

Figure 4: An example of a block of LLaMA-2-7B [78] weights going through the H.265 pipeline. The intra-prediction step generates a rough prediction of the entire block, making the residuals easy to code with the DCT transform.

the frame and removes high-frequency information that is less perceptible to human eyes. One popular technique in transform coding is the DCT transform [38], which is utilized in both H.265 and H.264. The DCT transform is based on that for an orthogonal basis matrix $\mathbf{B}$ and input $\mathbf{X}$, the encoding process can be represented as $\mathbf{Y} = \mathbf{XB}$, and the encoded $\mathbf{Y}$ can be decoded by $\mathbf{X} = \mathbf{YB}^{-1}$.

In tensor compression, however, if we view tensors as images, the emphasis on low-frequency signals through DCT transform may visually preserve similarity, but this does not translate to improved compression quality in terms of the accuracy of model-specific tasks (e.g., the quality of the generated sentences). Instead, transform coding is effective in tensor compression for a different reason — it mitigates the encoding difficulties caused by outliers in tensors [3, 4]. Outliers that are far away from the center distribution, sometimes even degrees of magnitude different from centered values, put conventional quantization and compression techniques in a dilemma that could either encode the outliers but leave the center distribution's encoding in low resolution or clip the outliers to better adapt to the range of the center distribution, but not both. Prior works [19, 85] showed that suppressing the outliers or decreasing the center quantization granularity both decreases the accuracy of LLMs.

However, transform coding solves this dilemma. In Figure 3, we show the effect of the DCT transform: (a) exemplifies a common tensor distribution where the central distribution is near-normal, but outliers exist at both tails. The DCT transform solves the challenge of encoding outliers; its output, as seen in (b), no longer contains outliers. A more concrete example is shown in the process from (c) to (d), where we can see that the value of 128 is an outlier in (c). The DCT transform addresses this by amortizing the difficulty of encoding the outlier value 128 to other values within the same block. This results in a matrix (d) containing no outliers and is much easier to encode in binary space.

Intra-Frame Prediction: Intra-frame prediction is another crucial component in modern video codecs. It is based on the simple fact that objects in the frames can be predicted or approximated through a few classes of patterns. For example, smooth areas of the frame can often be approximated by predicting the pixel values from neighboring pixels using a planar or DC (direct current) prediction mode. Edge areas, which are common in real-world images, can be predicted using directional modes that capture the orientation of edges. Although it is usually impossible to predict the pixels in a block with very high accuracy, residual encoding can be used to improve the quality further. As long as the prediction is close to the original block, the residuals will be small in size and much easier to encode compared to the raw values.

To our surprise, the intra-frame prediction works well for compressing tensors. We present an example in Figure 4. In (a), a block of a weight tensor is depicted as an image. We then performed H.265 encoding on this image as a frame, extracting the prediction from the intra-frame predictor of H.265 using the HEVC Test Model (HM) [33]. The predicted image is shown in (b), and the residual, which is the difference between the original block and the prediction, is shown in (c). We made three observations for applying intra-frame prediction for weight images. First, the original weight, when viewed as images, contains edges and planar blocks that are similar to real-world images due to the channel-wise distribution property, as shown in prior works [4, 42, 85]. The channel-wise distribution property means each value's distribution aligns with the corresponding channel, causing values close to each other to appear within the same channel, which visually looks like the edges of objects. Second, the intra-frame prediction mechanism can detect the channel-wise distributions and efficiently encode them using small-sized prediction states. Third, the residuals after the intra-frame prediction are much smaller in size compared to the original weight distribution and require much fewer states to properly encode. This, when used in conjunction with Transform coding and quantization as shown in Figure 4 (d), results in sparse and small coefficients that are very easy and efficient to encode using only a few bits.

Inter-Frame Motion Prediction Does not Work. While the Inter-Frame prediction, including the motion prediction, achieves great efficiency in compressing videos, based on our experiments, it does not work for compressing tensors. As shown in Figure 2 (b) (5) $\longrightarrow$ (6), enabling the inter-frame prediction stage does not help reduce the number of bits per value but rather increases it. This observation suggests there is little inter-frame pixel correlation and little inter-layer correlation of weights in LLMs.

3.2 LLM.265 Implementation

We implement LLM.265 on top of the PyTorch [2] framework. Modern GPUs contain specialized hardware codecs designed to encode and decode videos. LLM.265 utilizes NVENC and NVDEC, which are the hardware video encoders and decoders present on Nvidia GPUs [59]. As NVENC and NVDEC have a maximum limit on the height and width of a frame, we first partition each input tensor into multiple chunks, each corresponding to a frame. Since video codecs take 8-bit integers as input, the FP16 values in the tensor need to be first rounded to 8 bits using quantization before feeding to HEVC codec [4, 10], with only the Luma channel of the codec used. As mentioned in Figure 3.1, inter-frame compression is ineffective; therefore, LLM.265 enforces an intra-frame-only encoding by setting codec parameters.

Although LLM.265 takes advantage of the hardware encoders and decoders of Nvidia's GPUs, LLM.265 is still bottlenecked by the limited throughput of them. This is because high performance is not the key design goal of existing hardware codecs. Since people typically watch videos at resolutions lower than 4K, and at a framerate lower than 60 frames per second, the hardware codecs lack the incentive to support higher throughput. In §4 and §5, we first present how LLM.265 can be applied to inference and training to compress activations, weights and gradients. In §6, we analyze the performance of NVENC and NVDEC, followed by why and how we should build tensor-specialized codecs to further speed up training and inference.

4 Memory- and Communication-Efficient Inference Using LLM.265

In this section, we introduce LLM.265 as the first *data-independent* method for low-bit (i.e., $\leq$ 3 bits for weights) LLM compression. LLM.265 is versatile and accurate, and it is *outlier-free, calibration-free, and training-free. Data-independent* means that LLM.265 does not require any extra data to *fine-tune* or *calibrate* the model for better accuracy, making the accuracy of LLM.265 independent from the quality of the chosen dataset. Moreover, it eliminates the compute-expensive calibration and fine-tuning steps. *Outlier-free* means that LLM.265, unlike existing works such as LLM.Int8() [19], does not require maintaining hard-to-encode outliers as separate sparse matrices. This avoids the slow random memory access patterns associated with those outliers and simplifies the system design, as LLM.265 operates entirely on continuous memory chunks.

Compared to existing quantization techniques, these features offer LLM.265 significant improvements in both efficiency and robustness. Building upon the LLM.265 implementation described in §3, we aim to improve the memory and communication efficiency in LLM inference. Our goal is to run a LLaMA-3-70B [1] model with a 128k context length on four edge devices with only 8GB of memory. Achieving this challenging objective requires a general-purpose compression strategy that includes three critical steps:

Weight Compression. In §4.1, we show that LLM.265 can reduce the memory footprint of model weight by 5.5× while maintaining accuracy. Our weight compression enables us to run a LLaMA-3-70B model with only about 25GB of memory.

KV Cache Compression. In §4.2, we employ LLM.265 to compress the KV cache to 2.9 bits without degrading accuracy. This reduces the cache size for a 128k length context from 40GB to 7.2GB for the LLaMA-3-70B model.

Communication Compression. In §4.2, we distribute the model across four devices using pipeline parallelism and compress the activations between different stages using LLM.265. By reducing the bit-width to 3.5 bits, our method can speed up the communication by 4.5 times.

4.1 Weight Compression

For low-bit weight quantization methods, quantization-aware training (QAT) [47, 86] is computationally expensive due to the high training cost. Conversely, post-training quantization (PTQ) [25, 42] is more efficient but heavily depends on the calibration set, which can limit its generalization ability across diverse models and tasks [62]. As a result, neither QAT nor PTQ can be considered truly zero-shot methods since they involve a "fine-tuning" step, raising concerns about their efficiency and robustness in real-world applications.

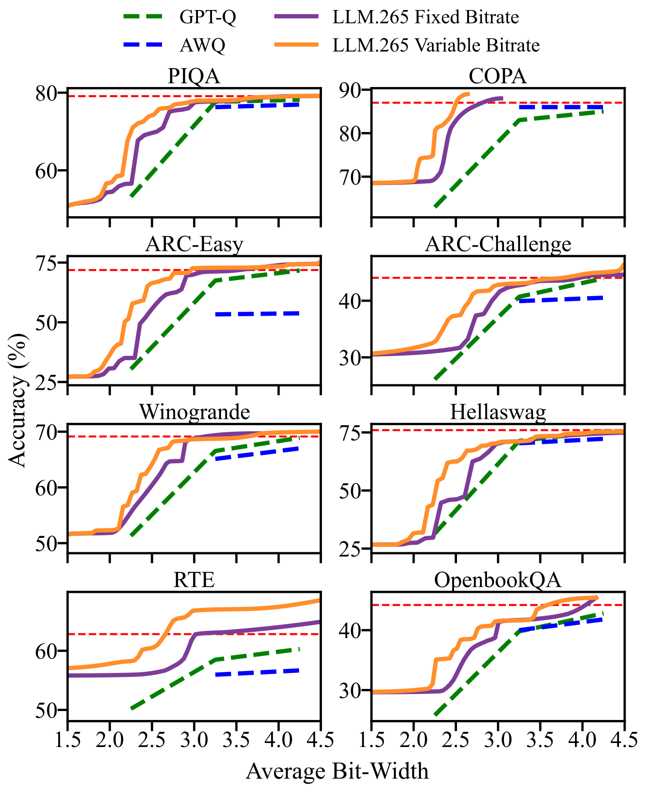

Figure 5: The trade-off between accuracy and average bid-width of different methods for compressing the LLaMA-2-7B model [78] on eight commonsense reasoning tasks. The red-dashed-line represents the BF16 uncompressed model accuracy.

As described in §3.2, NVENC only supports 8-bit integers as input, while the weights are stored in FP16 or BF16 precision. Therefore, we first quantize the input to 8 bits. In the second stage, we use video codecs to further compress the output from the first stage to a low bit-width (2-3 bits on average). A unique feature of LLM.265 is **variable and fractional bit-width compression**: by adjusting the video codec's parameters, we can control the compression budget, allowing the matrices to be compressed into fractional and variable bit-widths per element. This approach enhances both versatility and accuracy. Rather than quantizing the model to a fixed integer bit-width, we perform a fine-grained search[2] to maintain different compression ratios for different weight matrices.

We conducted experiments on the LLaMA-2-7B [78] and LLaMA-3-70B [1] models. Compared to SOTA quantization methods requiring calibration, LLM.265 achieves on-par accuracy with higher compression ratios.

Experimental Setup. Our evaluation of the proposed LLM.265 was carried out on eight zero-shot commonsense reasoning tasks

[2]In fixed-bit-width mode, the user specifies an average bit-width, and this budget is uniformly applied to all tensors. In variable-bit-width mode, the budget for each layer is defined as $B = k \cdot l + b$, where l is the layer index, k is a scalar we search for, and b is chosen so that the overall average bit-width matches the user-specified budget.

Table 1: Accuracy and average bits of LLaMA-3-70B [1] after compression using different algorithms evaluated on PIQA, WinoGrande (W.G.) and HellaSwag (H.S.) datasets. "128G" denotes quantizing each group of 128 values separately.

# Avg. Bits	Algorithm	PIQA	W.G.	H.S.
16	–	82.4	80.6	66.4
3.25	GPTQ-128G	80.6	77.1	63.5
	AWQ-128G	81.4	**78.6**	63.5
3.00	GPTQ	79.5	66.1	62.8
	AWQ	80.1	67.6	62.5
2.88	LLM.265 (**Ours**)	**81.5**	77.5	**63.7**

using the LM Evaluation Harness [27]. These tasks include PIQA [9], COPA [66], ARC-easy and ARC-challenge [13], WinoGrande [69], HellaSwag [87], RTE [81], and OpenbookQA [53]. For baselines, we compared LLM.265 with two state-of-the-art quantization methods: GPTQ [25] and AWQ [42]. As these baselines were calibrated using a few samples from WikiText-2 [52], we excluded the measurement on the calibration dataset from our experiments.

LLaMA-2-7B. In Figure 5, we compare LLM.265 and its fixed bitrate variant against other baselines. Our method significantly outperforms all baselines, maintaining full precision accuracy with approximately 3 bits. In contrast, GPTQ and AWQ achieve similar accuracy with around 4.25 bits. Additionally, these baselines struggle to keep accuracy under 3 bits, while LLM.265 generalize well to 2.5 bits. Another observation is that LLM.265 outperforms its fixed bitrate variant by a large margin in the extremely low bit-width regime (i.e., < 3 bits). This validates that different components in LLMs vary in their compression difficulty, and setting different bit widths can further push the limitations of compression.

LLaMA-3-70B. To verify the scalability of our method, we present the results of the compressed LLaMA-3-70B model on three datasets in Table 1. Our method achieves similar accuracy to GPTQ-128G and AWQ-128G with 0.37 fewer bits, and outperforms the 3-bit baselines without groupwise quantization with a large margin. Here, LLM.265 benefits from its fine-grained bit-width feature, reducing the bit-width to as low as 2.88 bits. In contrast, prior methods are limited to integer bit-widths with separate groups for quantization, making them less flexible.

4.1.1 Selection of Codecs. We also investigate how compressing using different video codecs affect model accuracy. Each Nvidia GPU generation supports a distinct set of codecs, summarized in Table 2. We exclude *VP9* [54] from our evaluation because it supports only hardware decoding, whereas our method requires hardware support for both encoding and decoding. We evaluate model accuracy under the remaining codecs; Figure 6 shows the average normalized accuracy across all tasks. For practical storage budgets ($>$ 1.8 bits per element) the curves for H.264, H.265, and AV1 overlap, with variations within the noise. Consequently, we adopt H.265 for GPU experiments, as it is the most widely available across GPU generations, supports the highest resolutions, and delivers the greatest throughput.

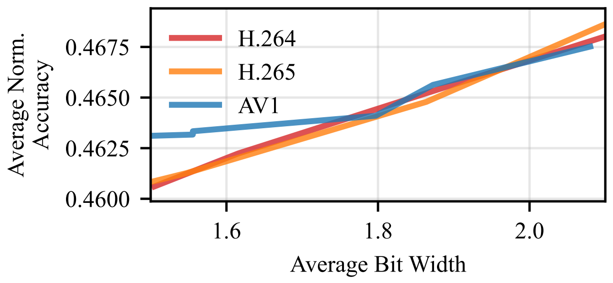

Figure 6: Comparison of the tensor compression information efficiency of the three most popular video codecs: H.264, H.265, and AV1. At a reasonable storage budget, the difference in information efficiency between all three codecs is so small that can be attributed to noise.

GPU Gen.	H.264	H.265	AV1	VP9
Ada Lovelace	4K Enc/Dec.	8K Enc/Dec.	8K Enc/Dec.	8K Dec
Ampere	4K Enc/Dec.	8K Enc/Dec.	-	8K Dec
Volta	4K Enc/Dec.	8K Enc/Dec.	-	8K Dec

Table 2: GPU Support for Different Video Codecs [14]

4.1.2 Applying LLM.265 on a Variety of Models and Tasks Other than LLM. We apply LLM.265 to a diverse set of downstream tasks; the results are presented in Figure 7. LLM.265 consistently surpasses standard quantization baselines and AWQ [42] across all tasks and model families, underscoring its strong generalizability.

4.2 KV Cache and Communication Compression

While our weight compression results show the ability of serving a 70B model on a single commodity-level GPU, it still suffers from two limitations described below.

Long-context scenarios: The large memory requirements of KV cache poses challenges [21, 74, 89] for long-context LLMs. For example, storing a 128k KV cache using FP16 requires 40 GB of GPU memory for the LLaMA-3-70B model, which is larger than the compressed model itself.

On-device Inference: It is infeasible to run inference for a 70B model on an edge device with only 8 GB memory. We address these challenges by applying LLM.265 to KV cache and communication compression, enabling distributed inference for LLMs in long-context, on-device scenarios.

Here, we detailed our final results that reduce the memory footprint by 5.5× and communication volumes by 4.5× for the LLaMA-3-70B model using LLM.265, which only lead to a minor accuracy drop ($<$ 2%) in the zero-shot reasoning task.

Building on our compressed LLaMA-3-70B model in §4.1, we further compress the KV cache to 2.9 bits and the activations between different pipeline parallelism stages to 3.5 bits. For baseline comparisons, we employ RTN quantization, SpinQuant [48] and QuaRot [4] to directly reduce the KV cache and activations to 3 bits and 4 bits, respectively, using asymmetric min-max dynamic

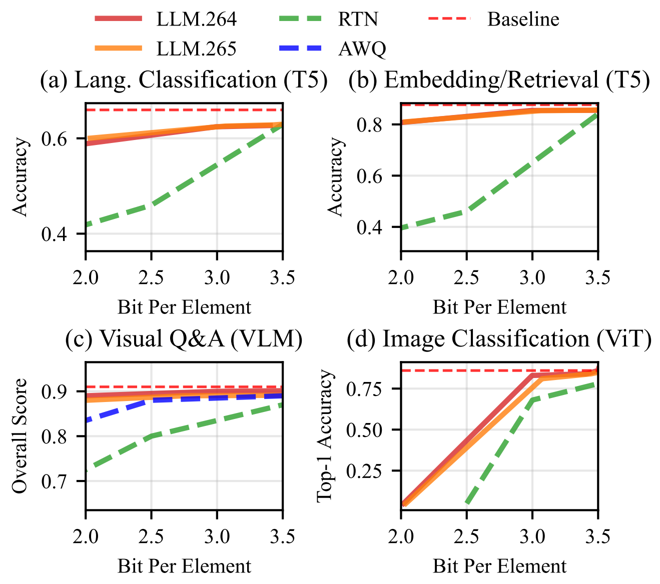

Figure 7: Evaluation of applying LLM.265 for model compression on four additional tasks: (a) Twitter Sentiment Analysis [50] using the T5 [58] model. (b) Quora Embedding/Retrieval [15] using T5 [58] model. (c) Visual Question Answering [46] using Qwen-2.5-VL-3B-Instruct model [6]. (d) ImageNet classification [18] using Vision Transformer (ViT) model [22].

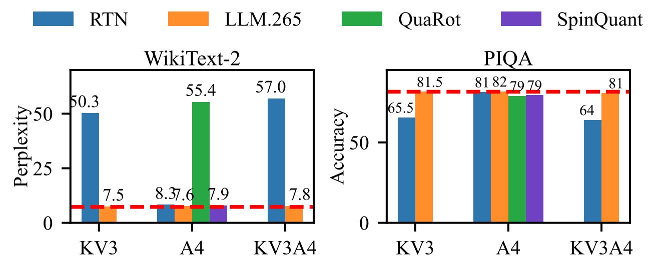

Figure 8: The comparison between RTN quantization, SpinQuant [48], QuaRot [4] and LLM.265 for compressing KV cache and activations of LLaMA-3-70B. "KV3" means compressing the KV cache to 3 bits (2.9 bits for LLM.265), while "A4" compresses activations to 4 bits (3.5 bits for LLM.265). Lower perplexity and higher accuracy indicate better compression quality.

quantization. Our evaluation includes measuring the perplexity score on the WikiText-2 test set and the zero-shot accuracy on the PIQA dataset. The perplexity score measures an LLM's fluency and coherence by quantifying its uncertainty in token prediction.

As shown in Figure 8[3], our method results in only a 7% increase (from 7.28 to 7.77) in perplexity score on WikiText-2 and a 1% drop (from 81.5% to 80.7%) in accuracy on PIQA while compressing the KV by 5.5 times and the activations by 4.5 times. Consequently, when the model is distributed across four devices using pipeline parallelism, only about 6.3 GB of memory is required for the compressed model and 1.8 GB for the stored KV cache. This amounts to approximately 8 GB of memory per device. Compared to our baselines, we observe that directly quantizing the KV cache to 3 bits leads to a significant accuracy drop, nearly destroying the original model's ability. For activation-only compression, our method achieves only a 5% increase in perplexity score while RTN quantization results in a 13% increase. These results show that LLM.265 consistently achieves higher accuracy than the baselines while using the same or less storage budget.

[3]Baseline accuracy results obtained from [76]

5 Communication-Efficient Distributed Training Using LLM.265

In this section, we shift to the distributed training setting, with the aim to reduce the communication costs, which account for 30% to 95% of the total training costs of modern LLMs [77, 82]. We demonstrate that LLM.265 effectively compresses various tensor types in two parallel training scenarios, showcasing its versatility and broad applicability.

5.1 Pipeline-parallel Training

We further demonstrate the effectiveness of LLM.265 in pipeline-parallel training, where our approach achieves significant compression ratios: 78% for activations and 37% for their gradients when communicating between pipeline stages.

Experimental Setup. We trained a 1.4B Pythia [8] model using 4-stage pipeline parallelism across 4 RTX 3090 GPUs, with compressed communication between distinct pipeline stages. Our training configuration used a sequence length of 2048, a micro-batch size of 4, and 8 gradient accumulation steps. We utilized a 5M-sample subset of the Pile dataset [26], reserving 5000 samples for validation and the rest for training.

Activation Compression. In Figure 9, we first verify the transfer of effective activation compression from inference (§4.2) to training using LLM.265(A), where we compress the activations to 3.5 bits. Compared to uncompressed training, LLM.265's activation compression is surprisingly beneficial. It not only reduces communication volume by 78% (from 16 bits to 3.5 bits) but also leads to faster convergence. This is evidenced by lower training loss and validation perplexity after 8K training steps (e.g., a validation perplexity of 24.1 compared to 42.7 for uncompressed training). We hypothesize that this improvement stems from LLM.265 acting as a denoising operation, filtering out noisy components in the activations and clipping the outliers in the corresponding weights' gradients during backpropagation [43, 88].

Gradient Compression. To further enhance communication efficiency, we compress the gradients of activations in LLM.265(A) + GQ and LLM.265(A+G), as illustrated in Figure 9. However, our experiments with LLM.265(A) + GQ reveal that gradients are more challenging to compress. Even directly applying a group-wise 8-bit RTN quantization to gradients proves ineffective, as the loss deviates from uncompressed training after only a few hundred steps. To address this issue, we introduce a residual compensation method for gradient compression. First, we compress the gradient G to approximately 3.5 bits, denoted as $\text{Comp}(G)$. Next, We further

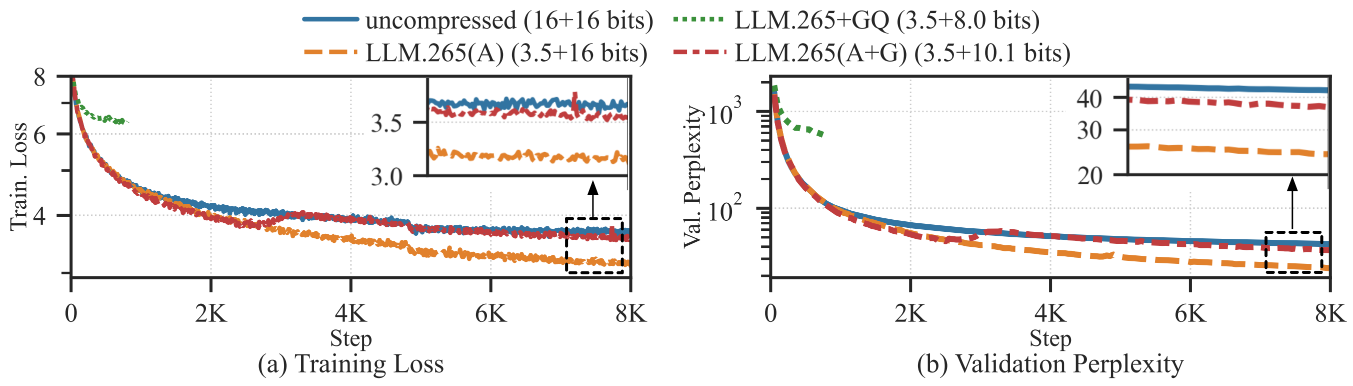

Figure 9: Training loss and validation perplexity of Pythia 1.4B using pipeline parallelism.

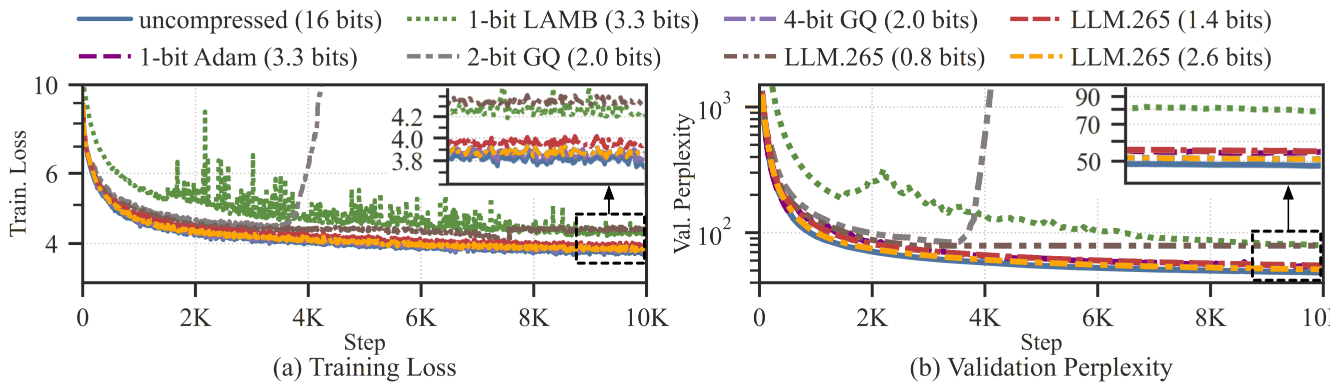

Figure 10: Training loss and validation perplexity of Pythia 160M using data parallelism.

compress the residual $G - \text{Comp}(G)$ using a two-stage strategy with different compression ratios: a) For the first 2500 steps, we use LLM.265 to compress the residual to 3.5 bits, achieving a loss curve similar to activation-only compression, and b) After 2500 steps, we switch to 8-bit RTN quantization for the residual. This two-stage approach is necessary because the training loss fails to continue decreasing after 2500 steps when using a 3.5-bit residual. This stagnation occurs because the range variance in gradients progressively increases from 1 to 3 orders of magnitude as training progresses, with some dimensions contributing significantly more to the loss. By employing this strategy, we achieve an average of 10.1 bits for the compressed gradient, calculated as $((3.5 + 3.5) * 2500 + (3.5 + 8) * 5500)/8000$. Still, an overall compression rate of 37% in gradient is achieved (from 16 bits to 10.1 bits) and the final validation perplexity is 36.7, which is lower than that of full-precision training.

5.2 Data-parallel training

Next, we show LLM.265 's ability to compress weight gradients communicated between GPUs to 1.4-2.6 bits from the starting of data-parallel training without modifying optimizers.

Experimental Setup. We trained the 160M Pythia model with a per-GPU batch size of 8. We adopt the same dataset as in §5.1.

Results and Analysis. In Figure 10, We first compare LLM.265 with state-of-the-art approaches to compress the gradients of weights in data parallelism: 1-bit Adam and 1-bit LAMB. Both baselines achieve an average bits of 3.25, as they require a warm-up period for the initial 15% of training iterations where gradients remain uncompressed. Empirically, 1-bit Adam achieves a validation perplexity of 54.6, while 1-bit LAMB reaches 79.0. LLM.265, however, achieves 51.0 with an average of only 2.6 bits, close to that of 48.2 for uncompressed training. To further explore the limits of our method, we introduce two variants with lower bit-widths. Our method can compress the gradients to 1.4 bits with a 54.8 perplexity, which is comparable to the best baseline using an average of 3.5 bits. When we further reduce the bit-width to 0.8 bits, our method converges early with a 78.7 validation perplexity, performing on par with the 1-bit LAMB baseline but at a much lower bit-width. This demonstrates the versatility of our method, as it can trade off between compression ratios and trained model quality across a wide range of bit-widths. Moreover, 1-bit Adam and 1-bit LAMB replace the widely adopted Adam optimizer, resulting in significant instability during training, as evidenced by large fluctuations in training loss. In contrast, LLM.265 does not make any assumptions about the training progress, eliminates the need for a warm-up period, and maintains stability throughout training.

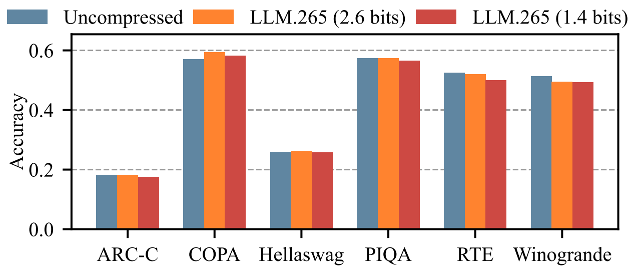

Figure 11: Evaluation results of trained Pythia 160M models on commonsense reasoning tasks.

In addition to these baselines, we also compare our method with 2-bit and 4-bit RTN quantization with a group size of 128. Our results shows that directly quantizing gradients to 4 bits results in a perplexity of 50.2, while the 2-bit variant completely fails to converge. The compression quality ranks as follows: LLM.265 (2.6 bits) > RTN (4 bits) > LLM.265 (1.4 bits) > LLM.265 (0.8 bits) > RTN (2 bits). Since RTN quantization is also a vanilla compression algorithm, these results further demonstrate the superior compression capability of LLM.265.

Model quality. To assess the quality of models trained with LLM.265, we compare their accuracy on downstream tasks with models trained without compression. The results are shown in Figure 11. We find that both LLM.265 (2.6 bits) and LLM.265 (1.4 bits) achieve comparable accuracy with the uncompressed model. LLM.265 (1.4 bits) maintains at least 95.2% of the uncomprssed model's accuracy, while for LLM.265 (2.6 bits), it is at least 96.6%.

6 Insights for LLM Accelerator Design

6.1 Limitations of NVENC/DEC

Despite LLM.265 achieving state-of-the-art information efficiency for compressing tensors, it is currently bottlenecked by the limited throughput of built-in video encoders and decoders on GPUs. Since people typically watch videos at resolutions lower than 4K and at frame rates lower than 60 frames per second, the video codecs on hardware such as the NVENC and NVDEC engines on Nvidia GPUs lack the incentive to support higher throughput. When used for tensor compression in LLM.265 , these engines limit the training and inference throughput. In our measurements, NVENC achieves a throughput of around 1100MB/s for compressing tensors, while NVDEC achieves a throughput of around 1300MB/s for decompressing video bitstreams to tensors, limiting the GPU's end-to-end communication bandwidth to 1100MB/s. Additionally, video codecs on existing GPUs only accept 8-bit inputs for video frames, requiring the compute cores to handle conversions from various precisions to 8-bit before executing the compression. This consumes compute resources that could otherwise be allocated for model computation, complicating system design.

In this section, we will delve deeper into the hardware implementation details of video codecs and propose augmentations for the design of future tensor-specialized codecs and compression-enabled training and serving systems. We found that video codecs are highly cost-efficient, with many of their components being redundant for tensor compression, offering opportunities for further optimization. We envision future accelerators could trade minimal cost to implement high-throughput tensor codecs, enabling more scalable and efficient distributed LLM training and inference.

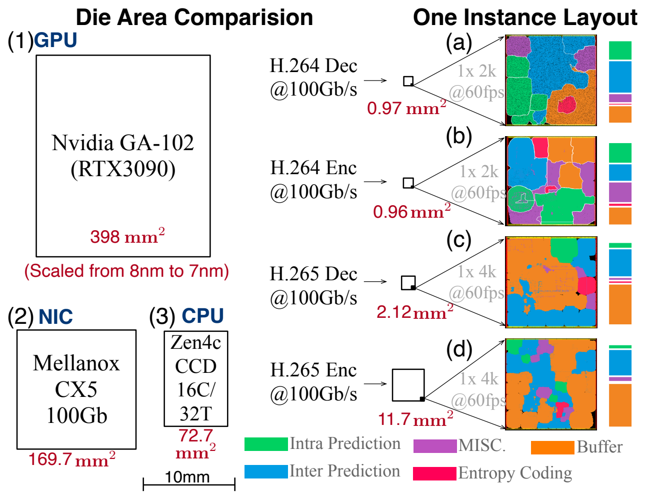

Figure 12: Comparison of the chip die area between GPU (1), CPU (3), NIC (2), and Video Codecs (a-d). Multiple instances of video encoders or decoders are combined to achieve a total throughput of 100Gb/s.

6.2 Die Area Analysis of Video Codecs and Other Devices in LLM Datacenters

Comparisons of the die area of video codecs with other devices commonly used in an LLM training data center, such as GPUs, NICs[4], and CPUs, are shown in Figure 12. To measure and model the hardware cost of these video codecs, we first obtained open-source RTL hardware implementations of both the encoder [23, 24] and the decoder [61, 71] for H.264 and H.265, respectively. We synthesized, placed, and routed these hardware modules using the ASAP7 [12] 7nm technology library. Note that a single instance of the codec supports resolutions up to 3840×2160 and a throughput of up to 60 frames per second. For fair comparisons, we normalized the throughput of the encoders and decoders to match the 100Gbps NIC bandwidth, thereby aggregating multiple instances of a video encoder or decoder in parallel to achieve the desired throughput.

Looking at the die area comparison of these devices, we observed that video codecs occupy significantly less space than other devices in data centers. The die area of an RTX3090 GPU is 628 mm^2 , which, when scaled down to 7nm [5], becomes 398m^2, while a combination of an H.264 encoder and decoder, each capable of processing up to 100 Gbps, requires less than 2 mm^2 of die area. This is 199× smaller than the GPU and 54× smaller than the Mellanox CX5 100Gbps Network Card. Therefore, **adding more codecs incurs only a negligible additional cost to the overall system while significantly enhancing both the effective memory capacity**

[4]The CX5 NIC die area was obtained by direct measurement, yielding dimensions of 12.14 mm × 13.98 mm = 169.7 mm^2.

Table 3: Energy for Communication versus Compression with Video Codecs and Customized Three-in-One Codecs.

	Power (W)	Area (mm^2)	Energy/Bit (pJ)
NCCL End to End	-	-	5120
H.264 Enc (100Gbps)	1.1	0.96	167.8
H.264 Dec (100Gbps)	1.0	0.97	154.3
H.265 Enc (100Gbps)	11.0	11.7	1707.5
H.265 Dec (100Gbps)	4.3	2.1	665.4
Three-in-one Enc (100Gbps T., 8K60fps V.)	**0.78**	**0.70**	**97.8**
Three-in-one Dec (100Gbps T., 8K60fps V.)	**0.58**	**0.58**	**63.5**

and the effective communication bandwidth, which are two of the most costly resources in the system.

As we have analyzed in §3, not all components of the video codec work well for tensor compression; specifically, inter-frame prediction, which includes motion prediction, significantly reduces the compression ratio. Figure 12 (a-d) presents the zoomed-in die layouts for the encoders and decoders, along with the die area distribution for each component. This distribution shows that a significant portion of the die area is spent on inter-frame prediction and the frame buffer. If the inter-frame prediction is removed, we save the die area spent for the inter-frame prediction logic and drastically decrease the buffer size requirement as frames no longer need to be maintained for analyzing inter-frame correlations. However, that will also make the codec no longer work for video encoding and decoding. In reality, the encoding and decoding of video streams remains a highly demanding workload on consumer-grade PC and mobile platforms, especially as video resolutions and frame rates continue to rise. These mobile and consumer-grade platforms often have limited memory capacity, memory bandwidth, and communication bandwidth while also facing strict constraints on energy efficiency and chip die area. Therefore, developing an effective tensor codec is essential for efficiently deploying larger and more advanced large language models (LLMs) on these platforms at the same time. The dual demand for efficient video codecs and tensor codecs makes it imperative to support both workloads on these platforms. Ideally, a codec should function efficiently for both applications, enabling hardware module reuse and achieving high performance and efficiency across tasks.

7 Enhance Video Codecs for Tensor Compression

In this section, we prototype and evaluate a novel three-in-one codec that supports the compression and decompression of tensors, videos, and images, as shown in Figure 13. We developed this codec using the H.264 video codec [23, 61] as a foundation. There are two main components of the H.264 codec: the first, illustrated in (b), is the pipeline that includes intra-frame prediction, which can be shared across all three input types. The second component, illustrated in (c), is specific to video processing and handles functions such as inter-frame prediction and motion estimation.

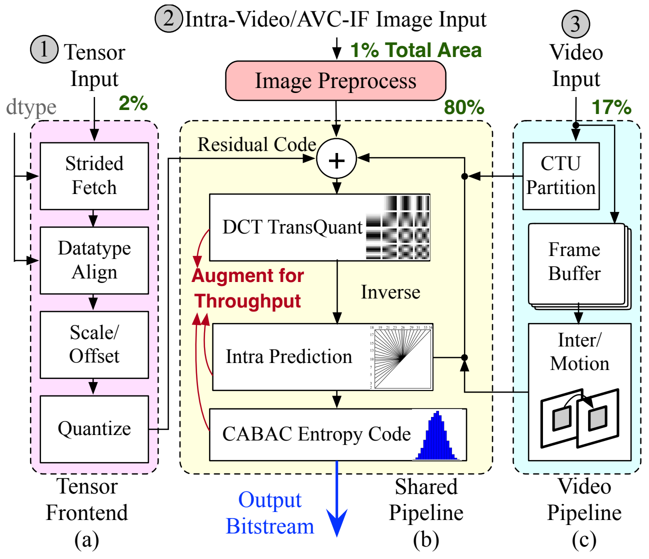

Figure 13: We modified the H.264 video codec to enable the encoding and decoding of tensors, images, and videos. The three-in-one encoder illustrated in this figure supports encoding tensors at a throughput of 100 Gbps and videos at 8K resolution and 60 fps.

The three-in-one codec design prioritizes the reuse and augmentation of the unified hardware compression pipeline depicted in (b), which is used for tensor inputs (Figure 13①), image inputs (②), and video inputs (③). Typically, video codecs are designed to handle up to 8K resolution at 60fps, while tensor codecs often require significantly higher bandwidth. To optimize resource allocation, our three-in-one codec dedicates resources to support 8K 60fps video encoding and decoding in the video pipeline (c) while augmenting the shared pipeline in (b) to sustain higher throughput at 100Gbps, enabling efficient processing across all input types. To support floating-point formats, we add a dedicated hardware block for data-type conversion and alignment (Figure 13(a)), offloading this work from the core compute units and improving efficiency. It is worth noting that the alignment unit also provides mixed-precision and micro-scaling support [67], matching the growing adoption of mixed precision.

During video encoding and decoding, pipelines (b) and (c) are active, while during tensor encoding and decoding, only (a) and (b) are utilized, with (c) remaining idle. When both multimedia and tensor-compression workloads are present, the shared pipeline is statically partitioned for both workloads by software. In our design, multimedia tasks receive higher priority because they are latency-sensitive, whereas tensor-compression workloads are primarily throughput-oriented therefore shared pipelines are allocated for tensors with lower priority. Remarkably, the shared pipeline accounts for 80% of the three-in-one encoder's total area, showcasing its reusability and efficiency compared to implementing separate codecs for video and tensor processing.

Additionally, we incorporated support for encoding images using the AVC Image Format [79]. This format essentially adapts the

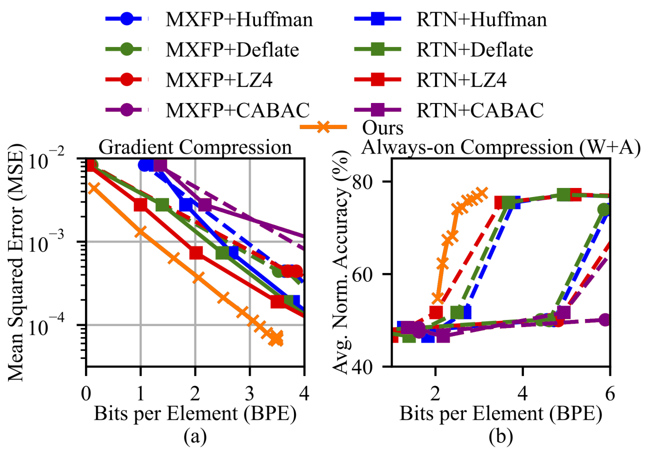

Figure 14: Information efficiency comparison with existing solutions: (a) Gradient compression MSE (lower is better) vs. average bit width. (b) Average normalized model accuracy (higher is better) across all evaluation datasets.

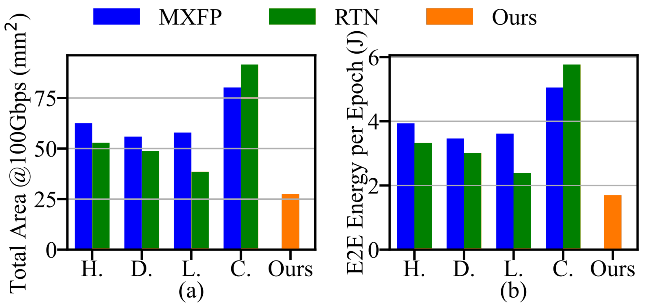

Figure 15: Comparing Three-in-one codec with existing solutions: (a) The total area of codec plus NIC required for achieving 100 Gbps end-to-end throughput. (b) The total energy of codec plus NIC for transferring the Pythia-125M gradient during one epoch. H., D., L., and C. represent Huffman, Deflate, LZ4, and CABAC algorithms.

H.264 codec by disabling all inter-frame compression features, aligning it closely with the tensor encoding workflow, and seamlessly integrating it into the three-in-one codec framework.

We implement and evaluate the three-in-one codec using the same flow as in §6.2. The design targets the ASAP-7 [12] technology library and is synthesized with Synopsys Design Compiler, followed by placement and routing in Synopsys IC Compiler II. Table 3 summarizes the power and area characteristics of the three-in-one codec implementation. The three-in-one codec demonstrates superior area and power efficiency compared to existing video codecs. The added support for video and image encoding introduces only marginal overhead, as most of the resources are allocated to the shared pipeline, which is effectively utilized across three input types. To evaluate our proposed three-in-one codec, in §7.1, we compare the three-in-one codec with existing approaches widely used for data and tensor compression. In §7.2 and §7.3, we focus on evaluating the performance impact and the scalability of deploying the three-in-one codec into LLM infrastructures.

7.1 Comparing Video Codecs with Other Compression Algorithms

Many existing works target the compression of various LLM tensors (weights, activations, gradients) to reduce memory and communication overhead. Two primary strategies involve introducing custom numeric formats (e.g., MXFP [28, 67]) and applying hardware-accelerated compression [40]; these can be used independently or combined to enhance overall compression ratios. Among them, microscaling floating-point (MXFP) formats and CABAC-based hardware compression [40] have shown particular promise due to their simplicity and demonstrated hardware efficiency. We implement these baseline approaches using a chained pipeline that's frequently used in tensor compression [10, 40]: first converting floating-point tensors to either MXFP or round to integer formats, then applying one of four compression algorithms (Huffman, Deflate, LZ4, or CABAC). This yields eight alternative "tensor codecs" (a 2×4 grid) that we evaluate alongside our three-in-one codec on the gradient, weight, and activation compression tasks.

Figure 14(a) plots the mean-absolute-error of gradient compression against bitrates. Under the same error budget, our three-in-one codec consistently uses fewer bits per element than any of the eight baselines, indicating superior information efficiency. We also evaluate always-on compression, where we compress both weights and activations during inference, using the same datasets that we used for evaluation in Figure 5. The result of the average normalized accuracy is shown in Figure 14(b). Again, our three-in-one codec maintains higher inference accuracy at lower bitrates than all baselines.

To evaluate hardware efficiency with variables controlled, we implement and synthesize these baseline codecs based on open-source RTL implementations [24, 40, 64, 71], all using the same technology library and synthesis settings as our three-in-one codec. Figure 15 reports the total area of the communication system (including both the NIC and the codec) required to sustain 100 Gbps effective bandwidth, along with end-to-end energy consumption for one epoch of Pythia-125M [8] gradient communication. Thanks to its higher information efficiency, our three-in-one codec achieves the best area and energy efficiency. This is not only due to the codec's low hardware cost and efficient hardware implementation but also due to its superior information efficiency, which allows the cost associated with the NIC, being the largest contributor to the overall area and power, to be greatly reduced by transferring less data.

7.2 Performance Impact of Compression

We developed an analytical model for modeling a distributed training cluster's performance and energy consumption to investigate the effect of enabling communication compression. The model takes the LLM's configuration and GPU specifications as input, including

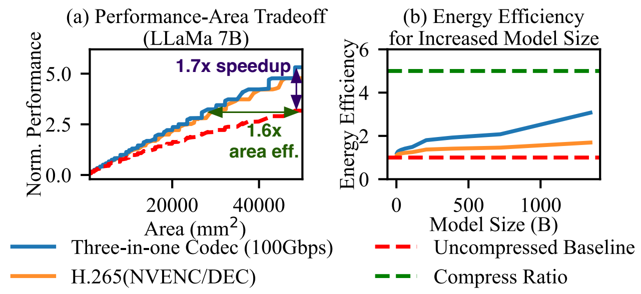

Figure 16: Impact of Communication Compression in Distributed LLM Training. The "Uncompressed Baseline" represents results without compression. The compress ratio determines the upper bound for speedup and energy efficiency.

memory capacity and GPU power. It then evaluates the performance of the training process and the power consumption of the GPUs, both with and without compression enabled. [5].

In our experiments, we tested the effect of compression on LLaMa 7B model training performance, as illustrated in Figure 16(a). We measured the cluster's performance by controlling the total die area budget, which could be allocated to GPUs, NICs, or codecs. We tested over 2,000 configurations of the hardware and training settings, including the number of GPUs, NICs, and codecs, as well as the data-parallel and pipeline-parallel ranks. We compared the performance and efficiency of three scenarios: 1) no communication compression, 2) communication compression using video codecs (NVENC/DEC), and 3) communication compression using our customized three-in-one codec. For each scenario, we plotted the Pareto frontier for area versus normalized performance. The results showed that scenarios with compression consistently and significantly improved performance under a fixed area budget. Furthermore, the three-in-one codec demonstrated superior performance and area efficiency compared to NVENC/DEC. When compared to the uncompressed scenario, the three-in-one codec achieved a 1.7x speedup over the uncompressed baseline at an area budget of 50,000 mm^2. On the other hand, to maintain a normalized performance of 2.6×, the three-in-one codec used 1.6× less total die area than the baseline, highlighting the substantial benefits of this codec for large-scale training.

7.3 Compression for Scalability and Energy Efficiency

As models increase in scale, communication bottlenecks become more severe due to the memory limitations of a single GPU, necessitating the division of the model into smaller parts for distribution across multiple GPUs. The communication bottleneck not only hampers training efficiency but also translates to a higher amount of energy spent on transferring data. We calculate the energy consumed for encoding/decoding one byte or transmitting one bit for codecs and interfaces in Table 3. When comparing the energy used for compressing and communicating, we observed that compression requires significantly less energy. For example, the combined energy used for the three-in-one codec's encoding and decoding is $\frac{5120}{97.8+63.5} = 31.7\times$ lower than that used for end-to-end communication with NCCL (as indicated by data from Table 3). As shown in §5, video codecs achieve a compression ratio of 3-20×. For example, if a 5× compression ratio on average can be achieved, it translates to $\frac{5120}{5120/5+97.8+63.5} = 4.32\times$ energy efficiency compared to transferring everything in an uncompressed format.

The modeling of compression-enabled training at the cluster level is shown in Figure 16 (b), where we plot the energy efficiency of using compressed communication using codecs versus increased model size. Communication power will account for a significant portion of the total power consumption for distributed LLM training. The larger the model is, the greater the percentage of power consumed by communication. By employing communication compression, the size of data being transmitted can be significantly reduced, resulting in power efficiency several times better than if left uncompressed. This highlights the importance of deploying high-bandwidth customized three-in-one codecs on GPUs and accelerators to ensure the scalability and sustainability of data centers for training future larger and larger LLMs.

8 Conclusion

LLM.265 repurposed video codecs as general-purpose and versatile tensor codecs. Leveraging the hardware video encoding/decoding engines available on modern GPUs, LLM.265 achieves state-of-the-art information efficiency for compressing weights, activations, and gradients of LLMs. This greatly reduces the pressure on the memory capacity and communication bandwidth of GPUs. To fully unlock the potential of LLM.265, we propose integrating specialized high throughput but cheap tensor codecs on future GPUs for more efficient distributed LLM training and inference.

Acknowledgments

This work was partially supported by NSF Athena AI Institute 2112562, NSF 2503010, NSF 2402696, NSF 2238665, and NSF 2504353. This work was also partially supported by a Google Research Award, a Google ML & System Junior Faculty Award, an Amazon Research Award, Intel, IARPA, Li Ideal, Moffett AI, and CMU CyLab Seed funding.

References

[1] AI@Meta. 2024. Llama 3 Model Card. https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md
[2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. *arXiv preprint arXiv:2312.06632* (2024).
[3] Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. arXiv:2401.15024 [cs.LG]
[4] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs. arXiv:2404.00456 [cs.LG]

[5]To calibrate the model with realistic data, we run micro-benchmarks on servers to verify the performance and measure the power used for communication. We measured the end-to-end communication power of NCCL [57] as indicated in Table 3. NCCL tests [56] was executed and power is measured using the power sensors on the Server Board Management Controllers (BMC). The effect on the performance of the communication systems are been verified against the Astra-SIM [84] LLM Infrastructure simulator.

[5] Scotten Jones at TechInsights. 2025. *Samsung 10nm 8nm and 7nm at VL-SIT.* https://semiwiki.com/semiconductor-manufacturers/samsung-foundry/7442-samsung-10nm-8nm-and-7nm-at-vlsit/

[6] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923* (2025).

[7] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems* 31 (2018).

[8] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*. PMLR, 2397–2430.

[9] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7432–7439.

[10] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2024. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems* 36 (2024).

[11] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[12] Lawrence T. Clark, Vinay Vashishtha, David M. Harris, Samuel Dietrich, and Zunyan Wang. 2017. Design flows and collateral for the ASAP7 7nm FinFET predictive process design kit. In *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*. 1–4. https://doi.org/10.1109/MSE.2017.7945071

[13] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).

[14] NVIDIA Corporation. 2025. *NVIDIA Video Codec SDK, version 13.0.* https://developer.nvidia.com/video-codec-sdk

[15] DataCanary, hilfialkaff, Lili Jiang, Meg Risdal, Nikhil Dandekar, and tomtung. 2017. Quora Question Pairs. https://kaggle.com/competitions/quora-question-pairs

[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. *Advances in neural information processing systems* 25 (2012).

[17] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[19] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. arXiv:2208.07339 [cs.LG]

[20] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]

[21] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. 2024. Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference. *arXiv preprint arXiv:2402.09398* (2024).

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[23] Yibo Fan. 2023. H.264 Video Encoder IP Core. https://github.com/openasic-org/xk264

[24] Yibo Fan. 2023. H.265 Video Encoder IP Core. https://github.com/openasic-org/xk265

[25] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG]

[26] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027* (2020).

[27] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation. https://doi.org/10.5281/zenodo.10256836

[28] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. arXiv:2208.14286 [cs.LG] https://arxiv.org/abs/2208.14286

[29] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[30] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[31] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).

[32] Nvidia Inc. 2024. Nemotron-4 340B Technical Report. (2024).

[33] International Telecommunication Union. 2023. *HEVC Test Model (HM).* Technical Report. https://hevc.hhi.fraunhofer.de/

[34] International Telecommunication Union. 2023. *ITU-T Recommendation H.264: Advanced Video Coding for Generic Audiovisual Services.* Technical Report. International Telecommunication Union. https://www.itu.int/rec/T-REC-H.264

[35] International Telecommunication Union. 2023. *ITU-T Recommendation H.265: High Efficiency Video Coding.* Technical Report. International Telecommunication Union. https://www.itu.int/rec/T-REC-H.265

[36] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[37] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23).* Association for Computing Machinery, New York, NY, USA, Article 82, 14 pages. https://doi.org/10.1145/3579371.3589350

[38] Syed Ali Khayam. 2003. The discrete cosine transform (DCT): theory and application. *Michigan State University* 114, 1 (2003), 31.

[39] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629* (2023).

[40] Alberto Delmas Lascorz, Mostafa Mahmoud, Ali Hadi Zadeh, Milos Nikolic, Kareem Ibrahim, Christina Giannoula, Ameer Abdelhadi, and Andreas Moshovos. 2024. Atalanta: A Bit is Worth a "Thousand" Tensor Values. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (<conf-loc>, <city>La Jolla</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(ASPLOS '24).* Association for Computing Machinery, New York, NY, USA, 85–102. https://doi.org/10.1145/3620665.3640356

[41] Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, and Yuxiong He. 2021. 1-bit LAMB: Communication Efficient Large-Scale Large-Batch

Training with LAMB's Convergence Speed. arXiv:2104.06069

[42] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv:2306.00978 [cs.CL]

[43] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[44] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024. QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving. arXiv:2405.04532

[45] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. 2023. LLM-FP4: 4-Bit Floating-Point Quantized Transformers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 592–605. https://aclanthology.org/2023.emnlp-main.39

[46] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. 2024. MMBench: Is Your Multi-modal Model an All-around Player? arXiv:2307.06281 [cs.CV] https://arxiv.org/abs/2307.06281

[47] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888* (2023).

[48] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024. SpinQuant–LLM quantization with learned rotations. *arXiv preprint arXiv:2405.16406* (2024).

[49] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173

[50] Maggie, Phil Culliton, and Wei Chen. 2020. Tweet Sentiment Extraction. https://kaggle.com/competitions/tweet-sentiment-extraction. Kaggle.

[51] D. Marpe, H. Schwarz, and T. Wiegand. 2003. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (2003), 620–636. https://doi.org/10.1109/TCSVT.2003.815173

[52] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]

[53] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789* (2018).

[54] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. 2013. The latest open-source video codec VP9-an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*. IEEE, 390–393.

[55] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1325–1334.

[56] nccl 2024. NCCL Tests. https://github.com/NVIDIA/nccl-tests.

[57] nccl 2024. The NVIDIA Collective Communication Library (NCCL). https://developer.nvidia.com/nccl.

[58] Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-T5: Scalable Sentence Encoders from Pre-trained Text-to-Text Models. arXiv:2108.08877 [cs.CL] https://arxiv.org/abs/2108.08877

[59] NVIDIA Corporation. 2024. NVIDIA Video Codec SDK. https://developer.nvidia.com/video-codec-sdk

[60] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774

[61] OsenLogic. 2024. Osen Loigc OSD10 H.264/AVC Baseline Video Decoder. https://github.com/ICscholar/H264_decoder-verilog-Cpp

[62] Davide Paglieri, Saurabh Dash, Tim Rocktäschel, and Jack Parker-Holder. 2024. Outliers and Calibration Sets have Diminishing Effect on Quantization of Modern LLMs. *arXiv preprint arXiv:2405.20835* (2024).

[63] Keivalya Pandya and Mehfuza Holia. 2023. Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations. *arXiv preprint arXiv:2310.05421* (2023).

[64] Gagandeep Panwar, Muhammad Laghari, David Bears, Yuqing Liu, Chandler Jearls, Esha Choukse, Kirk W. Cameron, Ali R. Butt, and Xun Jian. 2022. Translation-optimized Memory Compression for Capacity. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 992–1011. https://doi.org/10.1109/MICRO56248.2022.00073

[65] Stanislas Polu and Ilya Sutskever. 2020. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393* (2020).

[66] Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

[67] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. 2023. Microscaling Data Formats for Deep Learning. arXiv:2310.10537 [cs.LG] https://arxiv.org/abs/2310.10537

[68] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade

Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950

[69] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (2021), 99–106.

[70] Moritz Schubotz, Philipp Scharpf, Kaushal Dudhat, Yash Nagar, Felix Hamborg, and Bela Gipp. 2018. Introducing MathQA: a Math-Aware question answering system. *Information Discovery and Delivery* 46, 4 (2018), 214–224.

[71] Tianqi Shi. 2024. H265 decoder write in verilog, verified on Xilinx ZYNQ7035. https://github.com/tishi43/h265_decoder

[72] Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. 2023. Optimus-cc: Efficient large nlp model training with 3d parallelism aware communication compression. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 560–573.

[73] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191

[74] Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912* (2024).

[75] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).

[76] Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. 2025. FlatQuant: Flatness Matters for LLM Quantization. arXiv:2410.09426 [cs.CL] https://arxiv.org/abs/2410.09426

[77] Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 2021. 1-bit Adam: Communication Efficient Large-Scale Training with Adam's Convergence Speed. arXiv:2102.02888

[78] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[79] Trac D Tran, Lijie Liu, and Pankaj Topiwala. 2007. Performance comparison of leading image codecs: H. 264/AVC Intra, JPEG2000, and Microsoft HD Photo. In *Applications of Digital Image Processing XXX*, Vol. 6696. SPIE, 120–133.

[80] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396* (2024).

[81] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[82] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. {TopoOpt}: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.

[83] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[84] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 283–294. https://doi.org/10.1109/ispass57527.2023.00035

[85] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2024. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438 [cs.CL]

[86] Yuzhuang Xu, Xu Han, Zonghan Yang, Shuo Wang, Qingfu Zhu, Zhiyuan Liu, Weidong Liu, and Wanxiang Che. 2024. OneBit: Towards Extremely Low-bit Large Language Models. *arXiv preprint arXiv:2402.11295* (2024).

[87] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).

[88] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2019. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881* (2019).

[89] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems* 36 (2024).