

A Supplementary Analysis

A.1 Computation Requirements of TSLD

Training Method	QAT-KD Method	GPT2-0.3B 512		GPT2-0.8B 512		OPT-1.3B 256		OPT-1.3B 1024	
		Speed (iter/sec)	Memory (MiB)						
QAT	GT	2.03	19663	1.03	36317	5.15	10051	2.83	15167
QAT-KD	Logit	1.57	22622	0.81	40989	4.44	11589	2.27	17529
	GT+Logit	1.56	22622	0.81	40989	4.44	11589	2.27	17529
	L2L+Logit	1.51	31462	OOM	OOM	4.28	12143	2.12	25315
	TSLD	1.57	22622	0.81	40989	4.43	11589	2.26	17529

Table A1: QAT memory consumption and training speed study for KD method. The results are reported on the PTB dataset on the Ternary QAT-KD of GPT-2 and OPT series models with input sequence lengths ranging from 256 to 1024

The TSLD method integrates token-wise cross-entropy loss with Logit KD, involving two operations as detailed in Eq.6. Specifically, the term $\sum_{i=1}^V y_{n,i} \log(P_{n,i}^T)$ computes the cross-entropy loss from teacher logits (Z_n^T). This result, processed through a softmax function, derives the scaling value for each token. Multiplied element-wise with Logit KD term, $\sum_{i=1}^V P_{n,i}^T \log(P_{n,i}^S)$, it yields a token-wise scaled Logit KD. In fact, TSLD leverages the teacher logits that are pre-computed in Logit KD, circumventing extra memory usage. Furthermore, the associated computations have a complexity of $O(N)$, making TSLD’s overhead negligible for training.

To evaluate TSLD’s efficiency, we detail training speeds and GPU memory consumption for various QAT-KD methods using GPT-2 models in Table A1 left. Compared to Logit-based methods, TSLD maintains speed without extra memory consumption. In contrast, L2L KD stores intermediate activations from both the teacher and student models for KD, resulting in significantly increased memory requirements, evident from Table A1 left. As model size grows, as evidenced in scenarios utilizing GPT2-Large, memory requirements rise, leading to "Out of Memory" errors on an A100-40GB GPU. These findings highlight efficacy of TSLD, enhancing QAT-KD performance with memory comparable to Logit KD, while L2L KD demands significantly more. Even when the sequence length is extended from 256 to 1024, as Table A1 right shows, TSLD maintains the same GPU memory consumption and training speed as the Logit KD method.

A.2 Token Confidence Disparity Analysis

Our analysis of confidence disparity in token predictions, detailed in Section 4.2, extends beyond a specific GLM model. In fact, this observed trend is consistently present across various GLM models. As shown in Fig A1, we can distinctly observe the emergence of the *Low Confidence Region* (blue box) and the *High Confidence Region* (yellow box) consistently across models: OPT-6.7B(left), LLaMA-7B(middle), and LLaMA-2-7B(right). Additionally, as shown in Fig A1 right, we plot the token prediction’s statistics with varying input sequence lengths of 128 and 512. Regardless of the sequence length, the demarcation of confidence disparity remains consistent. This observation demonstrates that the TSLD methodology, grounded in the probabilistic dynamics of token prediction, can be universally applied across various GLMs.

A.3 Cumulative Quantization Error Analysis with LLM

In this section, we aim to expand our analysis of the cumulative quantization error discussed in Section 4.1 to GLMs larger than 6B parameters. By implementing 2-bit ternary quantization [30] on the OPT-6.7B and LLaMA-7B models, we assess the attention map quantization error in comparison to the FP model through MSE loss. These errors are visualized using a heatmap plot (Fig. A2 top), and the average attention map loss per token was plotted against each layer (Fig. A2 below). For the OPT-6.7B model, quantization error is measured for the 5th and 15th layers. Regarding the LLaMA-7B model, quantization errors are depicted for input sequence lengths of 128 and 512.

For the OPT-6.7B model at its 5th layer and the LLaMA-7B model with a sequence length of 128, we note an accumulation of quantization errors towards the latter tokens, as discussed in Section 4.1.

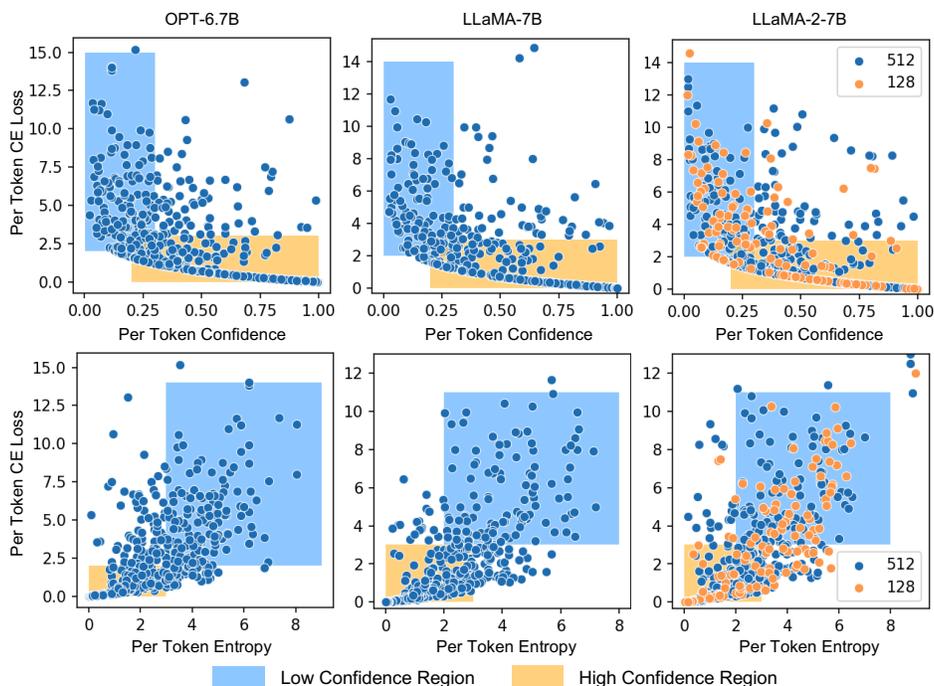


Figure A1: Scatter plots representing probabilistic relations of token prediction. The top plots show CE loss versus confidence for each token prediction, while the bottom plots plot CE loss with entropy. From left to right: OPT-6.7B, LLaMA-7B, and LLaMA-2-7B. For LLaMA-2, results for two input sequence lengths (128, 512) are plotted. Input dataset is wikitext-2

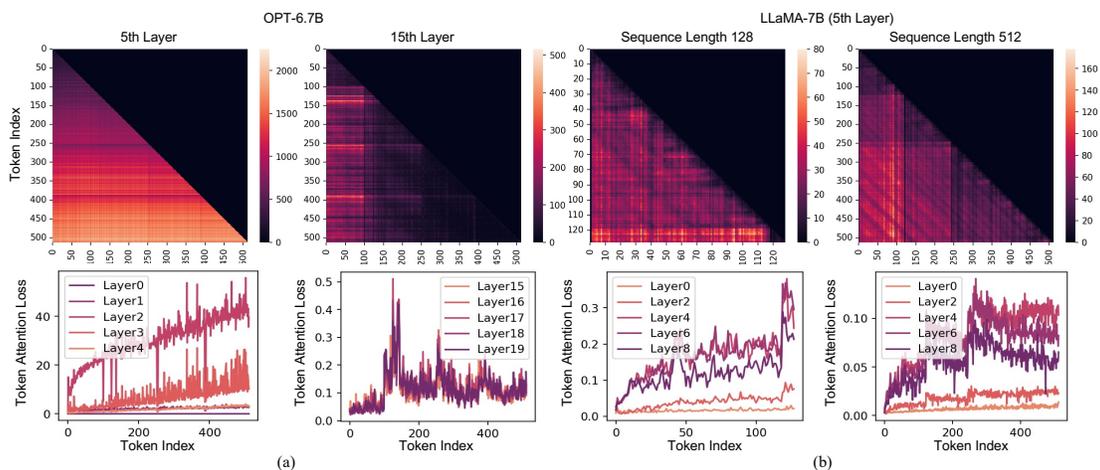


Figure A2: Top: Heat map of 2-bit ternary weight quantization error on attention map MSE loss. (a) OPT-6.7B's 5th and 15th layer attention loss. (b) LLaMA-7B attention loss for sequence lengths 128 to 512. Below: Average per-token attention MSE loss across layers from each attention map loss heatmap

However, as we delve deeper into the layers of OPT-6.7B or introduce longer input sequences to LLaMA-7B, this phenomenon becomes less pronounced. We speculate that this attenuation might arise from the intricate interplay of quantization errors as the depth of GLM increases, and the evolving attention patterns associated with varying sequence lengths influencing accumulation of quantization errors. A thorough exploration of cumulative quantization errors for larger GLMs will be reserved for future research.

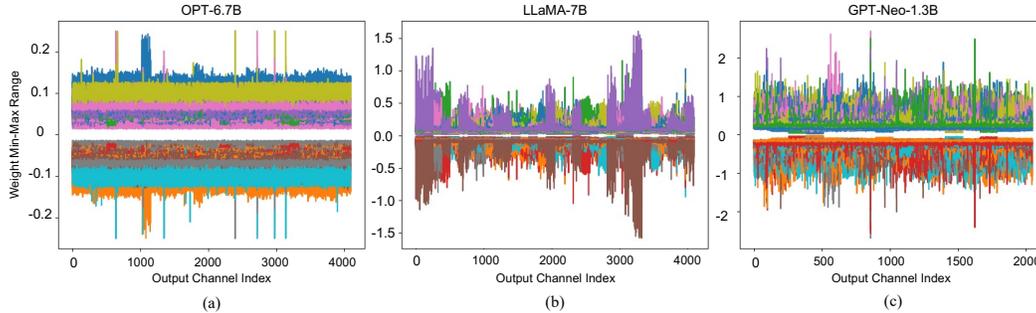


Figure A3: Min-Max range of Linear module weights for three types of GLMs per output channel. In the graphs, each color represents the min-max range of an each module weight. (a) OPT-6.7B (b) LLaMA-7B (c) GPT-Neo-1.3B.

A.4 Comparison of GLM Weight Distribution

Pre-trained GLMs show a wide variety of weight distributions [43]. We examine the Min-Max range of weights for each linear module across the output channel in various GLM models (OPT, LLaMA, and GPT-Neo) as visualized in Fig. A3. This analysis aims to elucidate the performance disparities observed in Section 5.5 due to quantization granularity (tensor-wise and channel-wise). For the OPT model, we observe that each module exhibits a consistent channel-wise min-max range, which is notably narrow, spanning from -0.2 to 0.2. In contrast, both LLaMA and GPT-Neo showcase a much more diverse min-max range across output-channels for each module, with the range itself being significantly broader, approximately from -2 to 2. This diversity in the output channel-specific min-max range clarifies the performance differences between tensor-wise and channel-wise approaches, as highlighted in Table 5. Specifically, OPT, which has limited output channel diversity, showed minimal performance differences between tensor-wise and channel-wise methods. Conversely, models like GPT-Neo and LLaMA, characterized by extensive channel diversity, exhibit significantly enhanced performance with channel-wise quantization. These findings suggest that determining the appropriate quantization granularity in QAT, with the aim of minimizing quantization error, necessitates a comprehensive understanding of the channel-wise weight distribution of the target GLM.

B Supplementary Experimental Results

B.1 8-bit Activation Quantization

Table A2 showcases experimental results applying both ternary weight quantization and 8-bit activation quantization (W2A8). We apply min-max quantization for activation quantization in the same way as in [19] [24] [25] [22], taking into account the asymmetric distribution of certain activation parts. Specifically, asymmetric min-max quantization is implemented in the multiplication of the Query and Key in self-attention and in the input activation of the FC2 linear layer³.

In W2A8, in line with observations from Section 5.2, L2L KD exhibits substantial accuracy degradation than Logit KD. Although Logit + GT performs less optimally than Logit KD due to the previously mentioned overfitting impact, our method outperforms the others across all model sizes, thereby underscoring the effectiveness of the TSLD method.

B.2 Clipping Value Exploration in 4-bit Weight Quantization

When adopting a QAT method like QuantGPT, which determines the clipping value with a learnable scale factor, it's crucial to initialize the scale factor appropriately to match the weight distribution of the quantized model. In the case of the OPT model, a much narrower distribution is observed compared to GPT-2, as illustrated in Fig. A4(a). If we set the clipping value ($\gamma=1.0$) in the same way as QuantGPT, we can observe that over 40% of weight elements are detrimentally clipped, as shown in Fig. A4(b). To alleviate the destructive clipping phenomenon in 4-bit quantization, we conduct an

³We use activation quantization code in the following repository <https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TernaryBERT>

Precision	Optimization Method	GPT				OPT			
		0.1B	0.3B	0.6B	1.5B	0.1B	1.3B	2.7B	6.7B
FP32 baseline		20.91	18.21	15.20	14.26	18.17	13.75	11.43	10.21
W2A8	L2L+Logit[25]	24.88	21.61	-	-	20.50	-	-	-
	Logit [24].	23.14	20.13	16.59	15.34	19.21	15.28	12.87	11.70
	Logit+GT.	24.37	20.78	18.01	16.87	21.59	16.58	13.49	12.81
	TSLD	22.01	18.83	16.26	15.23	18.92	14.95	12.14	11.43

Table A2: Impact of activation quantization (Ternary weight, 8-bit Activation Quantization results) in QAT-KD (tensor-wise)

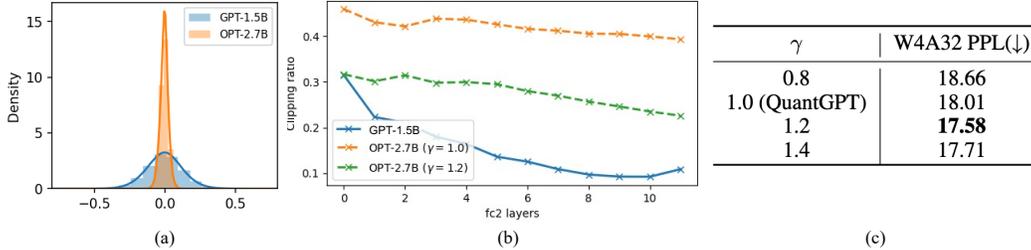


Figure A4: (a) Weight distribution of GPT-2-1.5B/OPT-2.7B (4th layer, FC-1) (b) We measure the ratio representing how many weight elements were clipped in the FFN-2 layers weight quantization. Upon applying QAT with the original QuantGPT recipe ($\gamma = 1.0$), we observe that over 40% of values were clipped in OPT-2.7B, a significantly higher rate compared to GPT-2-1.5B. (c) γ initialization exploration PPL results in OPT-0.1B with PTB dataset.

experiment exploring the initial value of the γ scale in QuantGPT. (γ scale determines clipping value in QuantGPT. Detailed quantization implementations of QuantGPT are further elaborated in C.2) Through exploration of γ initialization, we are able to reduce the proportion of clipped weights as in Fig. A4(b) by increasing the initial value of the γ scale, and consequently, achieving performance improvement as shown in Fig. A4(c). Through exploring γ scale hyper-parameters tailored to the OPT weight distribution, we manage to fairly compare multiple KD methodologies in 4-bit OPT QAT without the adverse effects of excessive quantization clipping. These experimental results suggest that, when initializing the clip value in the learnable clipping QAT method, one should consider the weight distribution characteristics of the target GLM.

B.3 Results of Decoder-Style BERT QAT

In Section 4.1, we discuss the cumulative quantization error due to the structural feature of the GLM’s masked self-attention, and compare the effectiveness of Logit KD and L2L KD in the QAT. In this experiment, we compare distillation methods in the Encoder model (BERT-base [27]), where, due to the absence of masking, the quantization error is evenly distributed among all tokens. According to [22], L2L KD is crucial in the Encoder model QAT KD, and having more layers to distill has proven beneficial for QAT performance.

As explained in Section 3.3, in the Natural Language Understanding tasks of the encoder model, we calculate the cross entropy loss using the representation of a single special token (class token, [CLS]) as logits. Drawing from the fact that the decoder model’s language modeling fine-tuning uses cross entropy loss of all token representations, we attempt to use every token’s final representation outputs as logits in the encoder model and measure cross entropy loss with the teacher model’s final token representation logits and use this loss as Logit KD (we call this KD method "Logit - All Token").

Task	RTE	STS-B	MRPC	CoLA
Full precision	73.28	89.24	87.77	58.04
Logit - [CLS] Token (Logit KD)	55.59	86.46	82.43	38.60
Logit - All Tokens	70.54	87.46	87.03	48.36
Logit - [CLS] Token + L2L (L2L KD)	72.34	88.98	87.70	51.12

Table A3: QAT-KD (tensor-wise) performance with multiple KD options on selected GLUE [39] tasks with BERT-base [27] model.

As can be seen in Table A3, the Logit - All Token method, which utilizes all final token representations as logits, is considerably more beneficial for performance than utilizing a single special token’s representation as Logit (Logit - [CLS] Token). However, when compared with Logit - [CLS] Token + L2L KD, we found that employing L2L KD yields superior performance in the QAT of the encoder model.

This additional experiment reveals that in an encoder model QAT KD where the quantization error is distributed among all tokens, L2L KD, which forces the student model’s each layer output to closely mimic that of the teacher model, is the most effective distillation method in QAT. This understanding extends our comprehension of how to adjust QAT KD methodologies to accommodate the structural nature of each model.

C Experimental Details

C.1 Model Description

Configuration	GPT				OPT				GPT-Neo	LLaMA
	0.1B	0.3B	0.6B	1.5B	0.1B	1.3B	2.7B	6.7B	1.3B	7B
# of Layers	12	24	36	48	12	24	32	32	24	32
# of Hidden Dim	768	1024	1280	1600	768	2048	2560	4096	2048	4096
# of Head	12	16	20	25	12	32	32	32	16	32
Learning Rate (FP)	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	5e-5	5e-5	1e-4	5e-5
Epoch (FP)	3	3	3	3	3	3	3	3	3	1
Learning Rate (QAT)	1e-4	5e-5	1e-4	7e-5						
Epoch (QAT)	90	60	30	30	90	30	30	10	30	5

Table A4: Configuration of each pre-trained decoder model with various sizes and hyper-parameter selection for fine-tuning FP and QAT-KD. All experiments consistently set a batch size of 4, and sequence length of 512 in language modeling fine-tuning

In our experiments, we conduct task specific fine-tuning for various pre-trained GLMs (GPT-2 [2], OPT [4]), GPT-Neo [40], and LLaMA [5]) at various sizes (0.1B to 7B). The GPT-2 pre-trained model has a vocabulary size (V) of 50257 and employs the GeLU activation function [44]. The OPT pre-trained model features a vocabulary size (V) of 50272 and uses the ReLU activation function [45]. On the other hand, the GPT-Neo pre-trained model has the same vocabulary size (V) as OPT and utilizes the new GeLU activation function [46]. It also incorporates Rotary Positional Embedding (RoPE) [47] for positional embeddings. As for the LLaMA pre-trained models, they have a vocabulary size (V) of 32000 and utilize the SwiGLU activation function [48]. These models also employ RoPE for positional embeddings. For detailed configuration information for each model size, please refer to Table A4.

C.2 Quantization Details

Quantization Aware Training with KD. In order to use KD in QAT, we need to initialize teacher and student models respectively. The teacher model undergoes task-specific fine-tuning in full precision (FP) based on a pre-trained model. The student model is then initialized from the teacher model, after which quantization is applied. The hyper-parameter settings of FP fine-tuning and QAT-KD, across various model types and sizes, can be found in Table A4. Furthermore, our experimental implementation utilizes the Huggingface language modeling code base⁴.

Post Training Quantization. We conduct our experiments of post training quantization with OPTQ and AWQ [14, 13], using the code from original paper respectively^{5 6}. We utilize a calibration dataset comprising 128 randomly selected 2048 token segments from the PTB [34] dataset for OPTQ and Pile [49] dataset for AWQ. To ensure a fair comparison with QAT, we adopt per-channel quantization as our quantization granularity.

⁴<https://github.com/huggingface/transformers/tree/main/examples/pytorch/language-modeling>

⁵<https://github.com/IST-DASLab/gptq>

⁶<https://github.com/mit-han-lab/llm-awq>

QuantGPT Implementation. In this paper, we primarily draw comparisons with QuantGPT, a state-of-the-art methodology above prior works regarding decoder QAT. This approach introduces two main contributions: a module-dependent scaling method and token-level contrastive distillation.

For 4-bit QAT-KD experiments, we adopt the QuantGPT [24] quantization method (module-dependent dynamic scaling). QuantGPT considers the quantization scale factor as a learnable parameter and optimizes it through QAT. Following the dynamic scaling method of QuantGPT, we determine the clipping value α for quantization by multiplying the average weight magnitude $\frac{\|\mathbf{w}\|_1}{n}$ with a learnable scale factor γ , where $\|\cdot\|_1$ denotes ℓ_1 norm: $\alpha = \gamma \cdot \frac{\|\mathbf{w}\|_1}{n}$. In this case, the initial value for the γ is set to 1, and the learning rate for γ is 0.0002.

Upon implementing token-level contrastive distillation, we observe issues of robustness in replicating the token-level contrastive distillation KD method, where incorrect choices in negative sampling could lead to performance degradation⁷. Therefore, To ensure a fair comparison, we exclude the contrastive loss from our implementation of Logit KD.

ALPACA-style Fine-Tuning for Arithmetic Reasoning Task. In arithmetic reasoning task (GSM8K) fine-tuning, We employ the ALPACA style fine-tuning method [12], proposed for instruction-following demonstration fine-tuning. This fine-tuning method fundamentally employs a language modeling approach, as demonstrated in Fig. 1(b), predicting the next word in a sequence. However, the ALPACA-style fine-tuning process has a distinctive characteristic: it transforms the datasets into a format that comprises instruction-response pairs, as illustrated in Table A5. We apply this ALPACA-style fine-tuning method to large pre-trained GLMs exceeding 2 billion parameters (OPT-2.7B/6.7B, LLaMA-7B).

D Examples of Arithmetic Reasoning Text Generation

In this section, we examine the QAT KD method on arithmetic reasoning task through a comparison of generation results from the QAT model. The GSM8K dataset serves as a benchmark for measuring arithmetic reasoning abilities, and models are expected to generate text responses auto-regressively based on the questions provided. This task requires not only correct mathematical calculations to produce the right answer, but also a logical problem-solving process, and the final answer is correct if both the logic and calculations are accurate. In evaluating GSM8K, we employed a greedy decoding strategy for the text generation process.

In Table A5, we can observe that the answers generated by the QuantGPT QAT model appear to make sense at first glance (corresponding to low PPL results in Table 2), but upon closer examination, it becomes evident that the necessary problem-solving process and computations are incorrect. Particularly in Question 1, the model writes that it should perform multiplication in the solution process, but actually executes division, leading to an incorrect intermediate result. From there, it continues to develop an entirely wrong solution. In Question 2, while the solution process and calculations align, incorrect methods are used to derive the intermediate results, eventually leading to a wrong answer. In Table A6, it can be seen that in Question 3, the model skips necessary intermediate steps in the problem-solving process, resulting in an incorrect answer. In Question 4, while the solution process is correct, there are errors in the calculation leading to an incorrect answer. In contrast, we can confirm that the generation results using our method accurately list the problem-solving process necessary to solve the question, just like the ground truth. Moreover, the calculation results are accurate, thereby resulting in the correct answer.

Such sample comparisons allow us to see the limitations of using Logit KD alone in reasoning tasks. As shown in Table A5 and A6, when Logit KD is used solely for QAT, as the low PPL result in Table 2 shows, it appears to well recover the natural text generation ability of the teacher model. However, in the results of the reasoning tasks that require a logical progression of solutions and accurate calculations, deficiencies such as errors in the solution process and calculation errors are observed. With our method that effectively combines GT Loss and Logit KD, we achieve the natural text generation capabilities of the teacher model while simultaneously enhancing logical reasoning and accurate calculation abilities through the ground truth during the QAT. Consequently, our method exhibits superior accuracy in the GSM8K benchmark.

⁷This issue has been acknowledged in the revised QuantGPT paper [50]

Question 1: *Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.* The zookeeper feeds all the apes in the zoo. He orders all the bananas from a local farm every 2 months. If the monkeys need 200 bananas, the gorillas need 400 bananas, and the baboons need 100 bananas every month, how many bananas does he need to order to last for 2 months? *Response:*

Ground Truth:

The total number of bananas the prime apes need every month is $200+400+100=700$.
Therefore in 2 months, the total number of bananas they need is $700 \times 2=1400$. ##### 1400

QuantGPT:

The zoo needs 200 bananas/monkey x 2 monkeys = 400 bananas for the gorillas.
It needs 400 bananas/gorilla x 4 gorillas = 1600 bananas for the baboons.
It needs 1600 bananas/baboon x 100 baboons = 16 bananas for the baboons.
It needs 16 bananas/baboon x 4 baboons = 64 bananas for the baboons.
It needs 64 bananas/baboon x 4 baboons = 256 bananas for the zoo.
It needs 256 bananas/2 months = 128 bananas to last for 2 months. ##### 128

Ours:

The zoo needs to buy 200 bananas/month x 2 months = 400 bananas.
The gorillas need 400 bananas/month x 2 months = 800 bananas.
The baboons need 100 bananas/month x 2 months = 200 bananas.
The total number of bananas needed is 400 bananas + 800 bananas + 200 bananas =
 $400+800+200=1400$ 1400 bananas. ##### 1400

Question 2: *Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.* The employees of Google went on a day trip. 4 buses were rented that have the capacity of holding 60 employees. 6 minibusses that can hold 30 employees, and 10 minivans that can hold 15 employees. How many employees can join the day trip? *Response:*

Ground Truth:

$60 \times 4 = 240$ 240 employees can ride on the bus.
 $30 \times 6 = 180$ 180 employees can ride on the minibusses.
 $15 \times 10 = 150$ 150 employees can ride on the minivans.
Therefore $240 + 180 + 150 = 570$ 570 employees can join the day trip. ##### 570

QuantGPT:

The capacity of each bus is 60 employees / 4 buses = 15 employees.
The capacity of each minibus is 30 employees / 6 minibus = 5 employees.
The capacity of each minivan is 15 employees / 10 minivans = 1.5 employees.
The total capacity of the buses and minibus is 15 employees + 5 employees + 1.5 employees = 20 employees.
The total number of employees who can join the day trip is 20 employees + 60 employees = 80 employees. ##### 80

Ours:

4 buses can hold $60 \times 4 = 240$ 240 employees.
6 minibusses can hold $30 \times 6 = 180$ 180 employees.
10 minivans can hold $15 \times 10 = 150$ 150 employees.
Therefore, $240 + 180 + 150 = 570$ 570 employees can join the day trip. ##### 570

Table A5: Samples of arithmetic reasoning outputs generated by OPT-6.7B ternary weight quantized with different methods. *Italics* part refers to the instruction formatting component in question.

Question 3: *Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.* On a particular week, a tow truck pulled ten cars for each of the first three days and then four fewer cars on each of the remaining days of the week. Calculate the total number of cars it towed that week. *Response:*

Ground Truth:

On the first three days, towing ten cars a day, the tow truck pulled $3 \times 10 = \langle 3 \times 10 = 30 \rangle 30$ cars.

It pulled four fewer cars each day, which is $10 - 4 = \langle 10 - 4 = 6 \rangle 6$ cars on the remaining days.

If it pulled ten cars for three days and four less on the remaining days, then it pulled for $7 - 3 = 4$ days, four cars less each day.

For the four days, the car pulled $6 \times 4 = \langle 6 \times 4 = 24 \rangle 24$ cars.

The total number of cars it pulled that week is $24 + 30 = \langle 24 + 30 = 54 \rangle 54$ cars.

54

QuantGPT:

On the first three days, the truck towed a total of $3 \times 10 = \langle 3 \times 10 = 30 \rangle 30$ cars.

On the remaining days of the week, it towed a total of $10 - 4 = \langle 10 - 4 = 6 \rangle 6$ cars.

The total number of cars it towed that week is $30 + 6 = \langle 30 + 6 = 36 \rangle 36$ cars.

36

Ours:

On the first three days, the truck pulled $10 \times 3 = \langle 10 \times 3 = 30 \rangle 30$ cars.

On the remaining days, it pulled $10 - 4 = \langle 10 - 4 = 6 \rangle 6$ cars each day.

The total number of cars it pulled on the remaining days is $6 \times 4 = \langle 6 \times 4 = 24 \rangle 24$

In total, the truck pulled $30 + 24 = \langle 30 + 24 = 54 \rangle 54$ cars that week.

54

Question 4: *Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.* Nick, Richard, Jason and DJ each have paintball guns. DJ has 8 guns, Nick has 10 guns, RJ has 1 gun and Richard has 5 guns. If they were to share their guns equally, how many guns would each of them have? *Response:*

Ground Truth:

The total number of guns is $8 + 10 + 1 + 5 = \langle 8 + 10 + 1 + 5 = 24 \rangle 24$ guns.

They would each have $24 / 4 = \langle 24 / 4 = 6 \rangle 6$ guns.

6

QuantGPT:

DJ has 8 guns, Nick has 10 guns, RJ has 1 gun and Richard has 5 guns.

For a total of $8 + 10 + 1 + 5 = \langle 8 + 10 + 1 + 5 = 22 \rangle 22$ guns.

If they were to share their guns equally, each of them would have $22 / 4 = \langle 22 / 4 = 5 \rangle 5$ guns.

5

Ours:

DJ has 8 guns, Nick has 10 guns, RJ has 1 gun Richard has 5 guns.

Total they have $8 + 10 + 1 + 5 = \langle 8 + 10 + 1 + 5 = 24 \rangle 24$ guns.

If they were to share them equally, each of them would get $24 / 4 = \langle 24 / 4 = 6 \rangle 6$ guns

6

Table A6: Samples of arithmetic reasoning outputs generated by OPT-6.7B ternary weight quantized with different methods. *Italics* part refers to the instruction formatting component in question.