To Reviewer 1: For question (1), the correlation among weights in kernels was indeed considered in the early design of MetaQuant: the meta gradient of weights is determined by its surrounding weights and the gradient of its quantized weight. However, such an implementation showed roughly 10-20% drop in performance compared to the current design. Besides, the kernel sizes differ across layers in a deep neural network: a meta quantizer that receives 3×3 kernel as inputs cannot be applied to the layers with 1×1 kernels. The independent process for each weight in the current design endows MetaQuant with generalization to arbitrary architectures of neural networks.

For question (2), we further tested training time per iteration as suggested for MetaQuant and DoReFa with STE using ResNet20 in CIFAR10 (Intel Xeon CPU E5-1650 with GeForce GTX 750 Ti). MetaQuant costs 51.15 seconds to finish one iteration of training while baseline method uses 38.17s. We will add this training time analysis in final version.

For question (3), we further conducted experiments and added some state-of-the-art results of binary / ternary network on ImageNet in Table 1:

Network	Method	bits	Top 1/5 drop (%)	Network	Method	bits	Top 1/5 drop (%)	Network	Method	bits	Top 1/5 drop (%)
ResNet18	MetaQuant	1	6.32/4.31	ResNet18	MetaQuant	2	5.17/3.59	MobileNetV2	MetaQuant	4	2.10/0.38
	STE*	1	7.70/5.43		STE	2	6.29/4.58		STE	4	3.71/1.89
	ELQ[1]**	1	3.55/2.65		TTQ[2]***	2(Ternary)	3.00/2.00				

Table 1: Comparison experiments in ImageNet.*: The baseline methods in paper that use dorefa as forward and STE as backward. **: ELQ is a combination of a series of previous quantization methods and tricks on incremental quantization. MetaQuant focuses more on how to improve STE-based training quantization, without any extra loss and training tricks. ***: TTQ is a non-symmetric ternarization with $\{0, \alpha, -\beta\}$ as ternary points. MetaQuant follows dorefa using a symmetric quantization which leads to efficient inference.

To Reviewer 2: Regarding "in eq. (8) the term $\tilde{\mathbf{W}}$ ...", we would like to clarify the computation order in a forward pass. In fact, for an iteration t, $\tilde{\mathbf{W}}_t$ is computed after ϕ as shown in the computation graph in Fig.2 of the paper. A more detailed illustration is shown in Fig.1: The meta quantizer $\mathcal{M}_\phi$ takes $\partial L/\partial \hat{\mathbf{W}}_{t-1}$ and $\tilde{\mathbf{W}}_{t-1}$ in the previous iteration as inputs to compute its output, parameterized by ϕ. The output of meta quantizer is then added to $\mathbf{W}_{t-1}$ to generate $\tilde{\mathbf{W}}_t$ in the current iteration t. Therefore, $\partial \tilde{\mathbf{W}}/\partial \phi$ can be computed if we track the path from ϕ to $\tilde{\mathbf{W}}_t$.

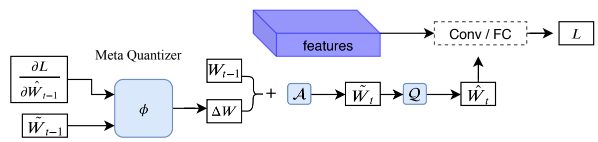

Figure 1: Learning process of the meta quantizer.

Regarding "... there seems to be a chicken-egg problem", the meta quantizer is actually linked to the final loss L of the base network with the following computation path: $\phi \rightarrow \Delta\mathbf{W} \rightarrow \tilde{\mathbf{W}} \rightarrow \hat{\mathbf{W}} \rightarrow L$, according to Fig.1. In detail: Step 1: In iteration t, $\partial L/\partial \hat{\mathbf{W}}_{t-1}$ and $\tilde{\mathbf{W}}_{t-1}$ (noted they are from the previous iteration) are fed into **meta quantizer** as data to generate meta gradient $\Delta\mathbf{W}$. Step 2: $\Delta\mathbf{W}$ contributes to $\tilde{\mathbf{W}}_t$, which is quantized to $\hat{\mathbf{W}}_t$. Step 3: $\hat{\mathbf{W}}_t$ is involved into convolution or fully connected operation with input features from the base network, finally leads to the **loss**. Intuitively, we can regard $\partial L/\partial \hat{\mathbf{W}}_{t-1}$ and $\tilde{\mathbf{W}}_{t-1}$ from previous iteration as data to meta quantizer (ϕ) for generating a component of $\hat{\mathbf{W}}_t$, and this process is differentiable. We can get $\partial L/\partial \hat{\mathbf{W}}_t$ using backpropagation, which can be passed to ϕ by chain rules.

Regarding "... should the loss function of the base network be used for training ...", note that the goal of base network is to minimize the final prediction loss while the aim of the meta quantizer is to provide accurate gradient $\partial L/\partial \hat{\mathbf{W}}$. Ideally, the meta quantizer should be trained using 'ground-truth gradients' as regression values. However, such 'ground-truth gradients' are inaccessible in practice. That's why STE is used to approximate the gradients in previous methods. In order to train the meta quantizer without ground-truth values, we instead treat the final prediction loss of the base network as indirect supervision. The final prediction loss could guide the meta quantizer towards reliable estimation for 'ground-truth gradients'. Therefore, in MetaQuant, the loss function of base network is used to train meta quantizer . Empirically, this indirect training shows better performance and faster convergence than STE in our experiments.

For the writing issues, thanks for pointing then out. We will correct them in the final version.

To Reviewer 3: Thanks for your comments. In fact, we conducted experiments in ImageNet in **Table.3 on page 7** in the original submission. Here, we have further conducted experiments, and added more state-of-the-art results of binary / ternary network on ImageNet in Table.1 for comparison.

References

[1] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9426–9435, 2018.

[2] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.