Thank you all for your time, feedback, and acknowledging the novelty and potential of our work. The comments significantly enhanced the manuscript and led us to stronger results. We address our major weaknesses below:
Clarity (R1-R4): We agree that a pseudo-code would have better clarified our method and its key points. The following pseudo-code for Node Koopman training has replaced our verbose explanation in Sec. 3. Pseudo-codes describing other implementations have also been added (Single Weight, Layer, etc. all follow from slight changes to Alg. 1).

Algorithm 1 Node Koopman training

Let $\mathbf{w}_j^l(t) \equiv [w_{j1}^l(t)\ w_{j2}^l(t)\ \dots\ w_{jN}^l(t)\ b_j^l(t)]^\dagger$ be weights/bias going **into** node j of layer l after t iterations of training. For each node (l, j), we wish to find an operator $\tilde{U}$ that satisfies $\mathbf{w}_j^l(t+1) = \tilde{U}\mathbf{w}_j^l(t)$. Choose $t_1 < t_2$, and $T > 0$. Train the NN of choice for t_2 training iterations and record all $\mathbf{w}_j^l(t)$ from t_1 to t_2. For each node (l, j):

1. Define matrices: $F = \left[\mathbf{w}_j^l(t_1)\ \ \mathbf{w}_j^l(t_1+1) \cdots \mathbf{w}_j^l(t_2-1)\right]$ & $F' = [\mathbf{w}_j^l(t_1+1)\ \ \mathbf{w}_j^l(t_1+2)\ \cdots\ \mathbf{w}_j^l(t_2)]$.
2. Compute $\tilde{U} = F^+F'$, where $F^+ = (F^\dagger F)^{-1}F^\dagger$.
3. Use $\tilde{U}$ iteratively to predict the evolution of $\mathbf{w}_j^l$ from training iteration t_2 to training iteration $t_2 + T$.

(R2, R4) Sec. 2 is broken into subsections for ease of reading and the Koopman mode decomposition is now discussed in the Conclusion. **(R4)** We correct a typo in Supplement Table S2 - $t_{\text{start}}, t_{\text{stop}}$ should be t_1, t_2. We agree that Fig. 2b and 2c overly complicate the discussion. They have been replaced by clearer text explanations, while Fig. 2e has been better labeled. T^{eq} is the number of training iterations needed by the standard optimizer to achieve the loss performance obtained by T iterations of Koopman training (both are with respect to t_2). A Koopman training attempt is considered successful if $T^{\text{eq}}/T > 0$, with values near 1 implying Koopman training accurately approximated the optimizer in terms of loss performance. Fig. 2e was meant to highlight that our original result of $T^{\text{eq}}/T = 0.99$ was due to our method's accuracy in predicting the evolution of individual weights/biases in training iteration time. The x-axis of Fig. 2e shows the true movement of individual weights/biases from t_2 to $t_2 + T$, while the y-axis represents the error in the predictions made via Node Koopman training (Alg. 1). **(R4)** Koopman training *exploits* existing weight/bias evolution information to predict their future states (or direction, in the case of the perceptron).

Generalization (R1-R4): We agree that our original examples are limited, although Alg. 1 shows that Koopman training is inherently data-driven and generally applicable to other choices of NN optimizer, size, and problem of interest. Given the novelty of Koopman training and claims of generalizability, we appreciate the need for more experiments. **(R3)** We do note that the Hamiltonian NN (HNN) is solving regression problems - their novelty is in the loss function choice. Further, a major issue for physics inspired NNs (indeed many NNs) is the performance in the latter half of training: our simple methods are already providing useful cost reductions in this regime.

(R1-R4) We replicated experiments with different choices of widths/depths, optimizers, and problems of interest - including a deeper feedforward NN trained on the full MNIST dataset (adapted from the official Pytorch MNIST example). We leveraged the platform agnostic nature of Koopman training by using MATLAB for implementation, which results in even larger speed ups, presumably due to faster/more robust matrix operations (as noted in our original manuscript, line 272). **(R2-R4)** Koopman training generalized well (Table 1). We re-emphasize that our current approach **does not** make use of parallelization (even though it is better suited, since only matrix calculations are involved). No fundamental/technical changes in our methods were made for the new experiments in comparison to the originals, other than those mentioned in Table 1. Complete details have been provided in the Supplement.

Table 1: Generalization results

Experiment	Optimizer	Architecture	KOT success	T^{eq}/T	Speed up
HNN	Adam	1:4:4:2	93%	0.99	103x
HNN	Adam	1:10:10:2	84%	0.85	58x
HNN	Adagrad	1:10:10:2	92%	1.00	62x
HNN	Adadelta	1:10:10:2	100%	0.98	64x
HNN	Adadelta	1:50:50:2	92%	1.00	17x
MNIST	Adadelta	784:10:10:10:10	100%	1.00	27x
MNIST	Adadelta	784:20:20:20:10	100%	2.00	37x

We hope that these additional comments and experiments enhance the clarity and perspective of our work. We apologize for any concerns we were not able to address in this response due to limitations of space and have ensured that every raised point has been addressed in the revised manuscript.