A Alternative Approaches for Quantifying Path Independence

We review alternative methods for quantifying how path-independent a given equilibrium model is, and why the Fixed Point Alignment score is the most suitable one amongst them for our analyses. We want a path independence metric to satisfy three criteria: 1) **Dimensionless:** Metrics computed from different training runs should be directly comparable with each other. To ensure this, one has to make sure that one uses dimensionless metrics (i.e. doesn't have units). 2) **local and global path independence:** The metric should test for path independence not just in a local neighborhood of fixed points, but over a sufficiently large portion of the initialization domain. 3) **efficiency:** It should be computationally tractable.

With these in mind, we review three alternative ways of quantifying path independence. The summary of the analysis is described in Table :

- **Jacobian Norm:** The Frobenius norm of the Jacobian of the output of an equilibrium model with respect to its fixed point initialization (i.e. $||\frac{\partial \text{FIX}(\boldsymbol{x}, \boldsymbol{z}_0)}{\partial \boldsymbol{z}_0}||_2$) gives a robust measure of how locally sensitive the network is to initializations. If this quantity is very small (and approaching 0 with more root finding steps), one can conclude that the given equilibrium model is path-independent on the given input. The main issues with this approach are: 1) It's extremely computationally intensive to compute (since both the input and the output of FIX are high-dimensional, neither forward nor backward differentiation can estimate this Jacobian efficiently) 2) It has units, meaning it isn't comparable across different training runs, 3) It only measures local path independence.
- **Agreement with Adversarial Initializations:** This method quantifies to what extent it is possible to directly optimize initializations such that they result in different fixed points. If the network is path independent, there shouldn't exist an adversary that can find adversarial initializations. The exact procedure for how adversarial fixed points can be found in Section B.4. The downsides with this approach is that 1) it's expensive - one has to solve an optimization problem for every problem instance, potentially using different optimizers and initializations, and 2) it only checks for local path independence.
- **Agreement with Swapped Initializations (Fixed Point Alignment):** The main idea behind this approach is to initialize the forward pass of an equilibrium model with fixed points obtained from other problem instances, and checking if the network still displays the same asymptotic behaviour (i.e. finds the same fixed point). If one uses cosine similarity to measure consistency, then one recovers Fixed Point Alignment score proposed in Section 4.2. Another option is to use the average Euclidean distance to check for consistency. This is identical to computing the trace of the covariance matrix of the distance between the computed fixed points. This approach – especially the one that utilizes cosine similarity to quantify consistency – is appealing, as 1) because it's unitless, it can be compared between training runs on the same task, 2) it checks for non-local convergence by sampling from a distribution of relevant fixed points 3) it's very efficient, requiring only two forward passes. The version that utilizes Euclidean distance does have units, hence cannot be used for cross-model comparisons [9]) To further make sure that this metric is correct, and does subsume with local path independence, we ran stress tests described in Section 4.2 to make sure this metric produces results consistent with the *agreement with adversarial initializations* method described above.

Alternatives to Cosine Similarity in Measuring Directional Alignment While cosine similarity is a conceptually clean way of quantifying directional alignment, one can consider alternative kernels as well. Ideally, our main results should not depend on the specific implementation details of path independence metrics (as long as they satisfy the criteria we set out above).

To check how sensitive our results are on the precise functional form of the AA score, we replaced it with three other kernels and re-assessed whether the results pointed to the same high-level takeaways. Concretely, we tried the Gaussian kernel $\phi_g(r) = \exp(-(\epsilon r)^2)$, Laplacian kernel $\phi_l(r) = \exp(-|\epsilon r|)$ and inverse multiquadratic kernel $\phi_i(r) = \frac{1}{\sqrt{1+(\epsilon r)^2}}$ where the input to the kernel function is

[9] If one uses LayerNorm [Ba et al., 2016] or similar normalization layers, we could expect the Euclidean distance based metric to behave similar to the cosine similarity based one.

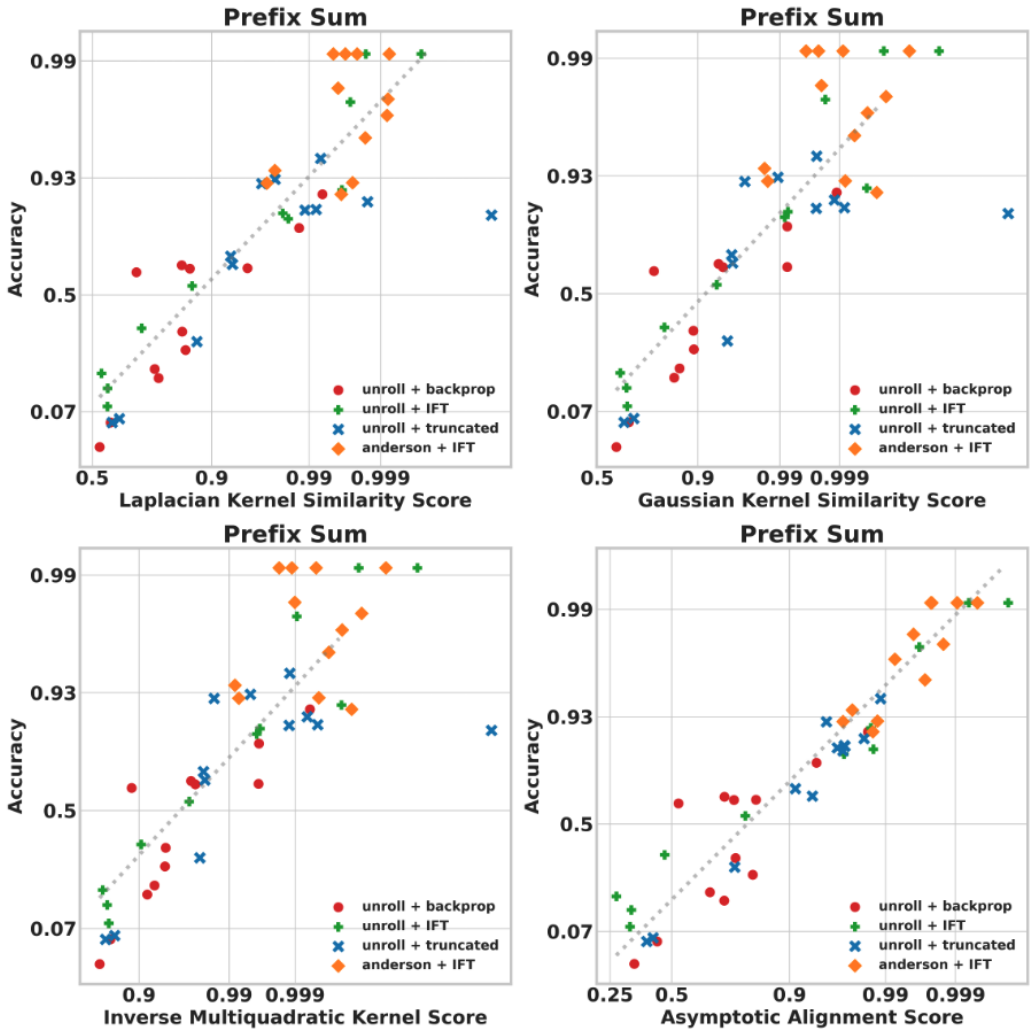

Figure 6: **Alternatives to Cosine Similarity in AA Score:** Replacing cosine similarity (right bottom) with alternative kernels like Gaussian (left top), Laplacian (right top) and inverse multiquadratic kernel yields qualitatively similar results. This demonstrates that the takeaways from our experiments don't depend on the particular implementation details of the ways path-independence is quantified.

the Euclidean distance of the normalized fixed points $r = ||\frac{z_1}{||z_1||} - \frac{z_2}{||z_2||}||$. We used $\epsilon = 5000$, which gave a reasonable dynamic range in the resulting values. The results can be seen in Figure 6. The takeaways from the plots remain identical: *path independence is strongly correlated with generalization, regardless of the specific details of how path independence is quantified* (as long as it satisfies the criteria we set in Appendix A).

Importance of using dimensionless metrics: Being unitless is necessary for a metric to be comparable across training runs, though obviously not sufficient. Consider two equilibrium models M1 and M2, where the fixed points computed by M2 have the same direction as those computed by M1, but have twice the Euclidean norm. The behaviour of this M2 is qualitatively the same as that of M1, but any metric that depends on the Euclidean metric (or Jacobian L2 norm, or any non-unitless metric) would report this network to be less path independent. Note that this is not a purely theoretical consideration: simply adding L2 regularization, or penalizing the magnitude of fixed points to encourage convergence will directly impact the scale of the fixed points (things that practitioners often use), thereby rendering non-unitless metrics unreliable.

Path Independence Metrics	Dimensionless	Local & Global Coverage	Efficient
IO Jacobian Norm	✗	✗	✗
Using Adversarial Initializations	✓✓	✓	✗
Fixed Point Alignment Score	✓✓	✓	✓✓

Table 3: **Comparing different ways of quantifying path independence:** We compare three different methods of quantifying how path independent a network is in terms of their correctness (whether they actually measure path independence), dimensionlessness (whether the quantity is unitless, allowing comparisons between between different networks meaningful), local and global coverage (whether the metric checks for path independence locally or globally) and efficiency (whether it's computationally cheap to compute the metric). Among the methods we've considered (See Section A), the Fixed Point Alignment Score is the most suited one for our purposes. Note that no metric considered here has perfect global coverage (i.e. can verify that a given network is globally path independent or not).

B Experimental Details

This section contains the additional information needed to reproduce our experimental findings. This section complements our code release.

B.1 Details for Experiments on Prefix Sum

Unless otherwise specified, all of the prefix-sum networks were trained under the following conditions:

Training details: We used the Adam [Kingma and Ba, 2014] optimizer and trained for 30000 gradient steps with a batch size of 150. We varied the initial learning rate on different runs (0.001, 0.0001 and 0.00001). We applied a stepwise learning rate decay both with a factor of 0.5 halfway and three-quarters through the learning. We also used gradient clipping at L2 norm equals 1.

Model selection: Since we didn't observe overfitting in our experiments, we used the last model saved model checkpoint during evaluation.

Training data: We used the length-32 training split of the prefix sum dataset described in Schwarzschild et al. [2021a].

Architecture: We used a 1D-convolutional depth-wise recurrent ResNet architecture roughly analogous to that used by Bansal et al. [2022] in their prefix sum experiments. The main differences include: (1) We used a single convolutional readout layer, as well as single convolutional layer to right before the depth-wise recurrent section of the network. (2) Unless otherwise specified, we initialized the fixed point at the 0 vector, unlike Bansal et al. [2022] who initialized the fixed point vector directly with the input. (3) While Bansal et al. [2022] uses concatenation to do input injection, we directly added the input (or a projection of it) to the hidden activations of the residual blocks, the same way the Kolter et al. [2020] does in their convolutional architecture. (4) We ablated using Weight Normalization [Salimans and Kingma, 2016], as this is something that has been reported to stabilize training.

Forward solvers: We used either fixed point iterations (of depth 6 or 32) or Anderson acceleration [Kolter et al., 2020] (with an iteration count of 6 or 32 and a memory of 3) depending on the experimental purpose of the experiment.

Backwards pass: We used either the standard backpropagation algorithm, the IFT gradients to train the networks or truncated backpropagation, depending on the purpose of the experiment. We used Anderson Acceleration to estimate the IFT gradients with an iteration budget of 10 and memory of 3. We found that rescaling the Jacobian term that appears in Equation 1 with a constant less than 1 stabilized training with the IFT gradients. We used a rescaling factor of 0.8. While this is relatively non-standard practice, we confirmed that using this stabilization strategy doesn't yield qualitatively different results than those reported in the paper. When we used truncated backpropagation, we dropped the gradients halfwah through the network (i.e. if the forward pass consisted of 10 iterations of the recurrent cell, we only backpropagated through the last 5 iterations).

Loss function: We used the cross entropy between the network predictions and the labels as the loss function. the prefix-sum task is a sequence-to-sequence problem, and each forward pass consists of multiple predictions. We used the average of all of the cross entropy values computed in a single forward pass.

Compute resources: All of the prefix sum networks were trained using Intel Xeon Silver CPUS and NVIDIA GPUs (both P100, T4V1 and T4V2 models were used in the experiments). Training one prefix-sum network took roughly 1 to 2 hours depending on how deep the forward pass is, and whether one is using IFT or backpropagation gradients.

B.1.1 Reproducing Table 1 (Stress-Testing the AA Score)

The non-path independent network used in this table is a 15-layer weight-tied input-injected network trained with backpropagation gradients. We set the maximum number of iterations for fixed point computation to 11 as we found that this choice gives the highest test accuracy on 64 bit prefix sum strings. While selecting the optimal value for maximum iterations, we increased the value up to 55 iterations. The path independent network is a 32-layer weight-tied input inejcted network trained with backpropagation gradients. The maximum iterations for this network are set to 500. We used a batch size of 1 and set the maximum number of L-BFGS [Liu and Nocedal, 1989] updates per batch to 50 for both the networks. We implemented this code in PyTorch and use these values of hyperparameters: `lr=1`, `tolerance_grad=1e-7`, `tolerance_change=1e-9`, `line_search_fn="strong_wolfe"`. We report AA scores and accuracy computed on 500 examples. We provide pseudocode in Algorithm Box 2

B.2 Details for Experiments on Mazes

Unless otherwise specified, all the networks for mazes were trained under the following conditions:

Training details: We used the Adam [Kingma and Ba, 2014] optimizer with an initial learning rate of 0.001 and batch size of 50. We use stepwise learning rate decay with a factor of 0.1 applied at epoch number 100. We trained all weight-tied input-injected networks for a total of 150 epochs. We observed that implicitly trained equilibrium models needed a longer training time for their validation accuracy to improve. All implicitly trained equilibrium models were pretrained for 150 epochs and trained with IFT for another 100 epochs.

Model selection: We used the checkpoint corresponding to the model with the best validation accuracy during evaluation. If multiple checkpoints had same validation accuracy, we selected the most recent checkpoint.

Training data: We trained on the training split of 9×9 mazes from the dataset described in Schwarzschild et al. [2021a].

Architecture: We used a 2D-convolutional depth-wise recurrent ResNet architecture analogous to that used by Bansal et al. [2022] in their mazes experiments. While pretraining implicitly trained equilibrium models, we do rollouts for 32-iterations. For equilibrium models trained with phantom gradient in Table 2, we use a damping factor of 0.75 and iterate for 2 iterations. Shallow weight-tied input-injected networks were trained with 8-layers while the deeper weight-tied input-injected networks used 32 layers. Both the feedforward networks are 32-layer deep non-weight tied networks. We ablate on input-injection for these networks. We used the original code released by [Bansal et al., 2022] to replicate the results for weight-tied input-injected networks trained with progressive training. In addition we ablate on weight normalization [Salimans and Kingma, 2016] for all the architectures—both unrolled and implicitly trained equilibrium models. We found that weight normalization helps stabilize the training, especially implicitly trained models. We found that using normalization layers (like group norm [Wu and He, 2018]) hurts generalization in mazes unlike prefix sum. Therefore, none of our architectures for mazes use any normalization layers.

Forward solvers: We used either fixed point iterations (of depth 8 or 32) or Broyden's method [Broyden, 1965] (with an iteration count of 40) depending on the experimental purpose of the experiment.

Backward pass: We used the standard backpropagation algorithm, the IFT gradients, truncated backpropagation or phantom gradients to train the networks, depending on the purpose of the experiment. We used Broyden's method to estimate the IFT gradients with an iteration budget of 40. For experiments with truncated backpropagation, we follow the exact setting as specified in [Bansal et al., 2022].

Loss function: We used the cross entropy between the network predictions and the labels as the loss function.

Compute resources: All networks for mazes were trained using Intel Xeon Silver CPUs and NVIDIA GPUs (1080Ti, 2080Ti and A6000 models were used in the experiments). Training a weight-tied network took roughly around 12-18 hours depending on how deep the forward pass is, whereas the implicitly trained models took around 2 days of training time on a single GPU.

B.3 Reproducing Figure 1

Both models were trained using 32 fixed point iterations and backpropagation. The path independent model is input injected, the path dependent model is not input injected. We run the network for 32 fixed point iterations on five different validation examples of length 32 to obtain different hidden layer initializations. We then run the models for 128 steps using these initializations to obtain five different trajectories. For the input injected network, we use the same input injection for all trajectories, obtained from a different example of length 32. We project the resulting hidden states onto two random orthogonal directions in the hidden layer state space.

B.4 Experimental Details for Section 3 (Upwards Generalization with Equilibrium Models)

B.4.1 Reproducing Figure 2a (OOD Generalization with Mazes)

The training details for all the weight-tied input-injected networks used to produce this plot can be found in Section B.2. The feedforward network used in this plot is a 32-layer non-weight tied network without any input injection that was trained with regular backpropagation. The maximum number of test-time iterations (both unroll and Broyden solver) for mazes of size 9, 13, 25, 31 and 59 were 50, 100, 400, 600, and 800 respectively. The test accuracy for mazes of a given dimension was computed on the entire split of $10,000$ mazes as provided in the original dataset [Schwarzschild et al., 2021b].

B.4.2 Reproducing Figure 2b (PI networks get better with more iterations)

The training details for the 12 networks used to produce this plot can be found in Section B.1. We varied (1) the depth (6 vs. 32), whether weight normalization[Salimans and Kingma, 2016] was used or not, and the initial learning rate (0.001, 0.0001 or 0.00001).

We evaluated each model on 128 examples of length 16, 32, 64, and 128 each, using fixed point iterations and measuring the average number of correct bits at each iteration. For each model and iteration step, we plot average accuracies over all splits and examples. The Fixed Point Alignment Scores for each model are calculated as in Figure 4a (see Appendix B.5.1).

B.4.3 Reproducing Table 1 (Stress-Testing the AA Score)

The non-path independent network used in this table is a 8-layer weight-tied input-injected network trained with backpropagation gradients. We set the maximum number of iterations for fixed point computation to 12 as we found that this choice gives the highest test accuracy on 13×13 mazes. While selecting the optimal value for maximum iterations, we increased the value up to 60 iterations. The path independent network is a 32-layer weight-tied input injected network trained with backpropagation gradients. The maximum iterations for this network are set to 500. We used a batch size of 1 and set the maximum number of L-BFGS [Liu and Nocedal, 1989] updates per batch to 50 for both the networks. We implemented this code in PyTorch and use these values of hyperparameters: `lr=1`, `tolerance_grad=1e-7`, `tolerance_change=1e-9`, `line_search_fn="strong_wolfe"`. We report AA scores and accuracy computed on 500 examples. We provide pseudocode in Algorithm Box 2

Algorithm 2 Stress testing AA Scores: An algorithm to find adversarial initializations

Input: A trained network $f_{\boldsymbol{w}}$, an input $\boldsymbol{x}$, N max number of LBFGS updates
Define: $h(\boldsymbol{y}_1, \boldsymbol{y}_2) := \frac{\boldsymbol{y}_1}{\|\boldsymbol{y}_1\|_2} \cdot \frac{\boldsymbol{y}_2}{\|\boldsymbol{y}_2\|_2}$
random init($\cdot$): A method that returns a random vector initialized from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
Initialize: $\boldsymbol{z} = \boldsymbol{x}$ or random init($\boldsymbol{x}$)
▷ Disable gradients
$\boldsymbol{z}_1 = \text{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}, \boldsymbol{x})$
▷ Clone $\boldsymbol{z}_1$ and Enable gradients on $\boldsymbol{z}_1$
for trials from 1 to N **do**
  $\boldsymbol{z}_2 = \text{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}_1, \boldsymbol{x})$
  Perform L-BFGS update on $\boldsymbol{z}_1$ to minimize $h(\boldsymbol{z}_1, \boldsymbol{z}_2)$
end for
Output: $\boldsymbol{z}_1$

B.5 Experimental Details for Section 5 (Path Independence Correlates with Upward Generalization)

B.5.1 Reproducing Figure 4a (Path Independence - Accuracy Correlation on Prefix Sum)

The training details for the 12 networks used to produce this plot can be found in Section B.1. We trained 12 networks under each category reported in the plot, where (1) the depth (6 vs. 32), whether weight normalization[Salimans and Kingma, 2016] was used or not, and the initial learning rate (0.001, 0.0001 or 0.00001) were varied.

Each value on the plots were computed using 300 examples from the validation splits corresponding to lengths 16, 32, 64, 128 and 256 (i.e. $5 * 300 = 1500$ samples per point). The average cosine similarities involved in computing the Fixed Point Alignment scores were computed using 10 different initialization per example. For the test-time forward pass, we used 512 iterations of Anderson Acceleration [Kolter et al., 2020] (note that this is using a more sophisticated solver using 16 times more forward iterations than the deepest of the trained networks). The trend lines on the plots were computed using the line that minimizes the least squares error.

B.5.2 Reproducing Figure 4b (Path Independence - Accuracy Correlation on Mazes)

The architectural and training details for all the networks in this plot can be found in Section B.2. Each value in this plot was computed using 500 examples from the validation splot corresponding to mazes of size 13 and 25. The average cosine similarities involved in computing the Fixed Point Alignment scores were computed using 10 different initialization per example. Unrolled + backprop networks use 100 test-time iterations for the forward pass for mazes of size 13×13, and 400 test-time iterations for mazes of size 25×25. We set the maximum number of permissible iterations for the Broyden solver [Broyden, 1965], and for non-weight tied feedforward network to the same values. The shallow unrolled + backprop network was trained using 8 layers. Empirically, we obtained the highest test accuracies on 11^{th} and 14^{th} iterations for mazes of size 13×13 and 25×25 respectively, for networks that use weight normalization. For networks that do not use weight normalization, the corresponding numbers are reported on 12^{th} and 18^{th} iteration. The trend lines on the plots were computed using the line that minimizes the least squares error.

B.6 Experimental Details for Section 6 (Experimental Manipulations of Path Independence)

B.6.1 Reproducing Figure 3 (Manipulations of PI)

The training details for the 12 networks that formed the baseline for the interventions are described in Section B.1. We trained 12 networks under each category reported in the plot, where (1) the depth (6 vs. 32), whether weight normalization[Salimans and Kingma, 2016] was used or not, and the initial learning rate (0.001, 0.0001 or 0.00001) were varied.

For each intervention, we kept all of the aforementioned training conditions except for the experimental manipulations that are intended to promote or hurt path independence.

- **Mixed initialization:** During each training forward pass, each sample was assigned with either zero initialization (i.e. the fixed point was initialized with the 0 vector) or standard normal distribution (i.e. ...) using a Bernoulli random variable of probability 0.5 (i.e. the examples that were run with zero vs. normal initializations were roughly half-half.
- **Randomized depth:** Randomized training was done by first specifying the minimum and maximum possible depths, then uniformly sampling a depth from that range during each forward pass. We used a range of 3 to 63 forward iterations for training 6 of the networks, and used a range of 32 to 64 for training the other 6.
- **Alignment penalty:** The alignment penalty, which was added to the original training loss in form of a regularized was computed as follows: (1) For each example in the batch, compute the fixed points k times using normal initialization, (2) Compute the dot product between each pair of fixed points (which yields $k^2 - k$ dot products, excluding each fixed point's dot product with itself), return the sum of the dot products, normalized by $k^2 - k$. In our experiments, we picked $k = 3$.

B.7 Experimental Details for Section 7 (Disambiguating Convergence and Path Independence)

B.7.1 Reproducing Table 2 (Training Convergence vs. Accuracy)

The architectural and training details for equilibrium models trained with phantom gradients and with IFT have been provided in Section B.2. Both the networks use Broyden solver for the forward pass. We set the number of maximum permissible iterations to 40 for in-distribution examples (*i.e.,* 9×9 mazes) and to 400 for 25×25 mazes. All the values reported in this table have been computed on 500 examples from the respective splits for in-distribution and out-of-distribution data.

B.7.2 Reproducing Figure 5a (Test Time Convergence vs. Path-Independence)

The architectural and training details of weight-tied input-injected networks used in this figure have been provided in Section B.2. The reported numbers have been averaged over 10 different runs. Each run uses a batch size of 100. The maximum number of forward iterations is set to 200 for both fixed-point and Broyden solvers.

B.7.3 Reproducing Figure 5b (Per Instance Path Independence vs. Accuracy)

To compute the histogram, we took the 12 networks trained with the training condition that yielded the strongest OOD accuracy (mixed initialization) and computed both the accuracy and AA scores on 300 examples for the prefix-sum dataset validation splits of lengths 16, 32, 64, 128 and 256. We grouped each AA score based on whether the prediction it's paired with was correct or not, then produced the histogram.

C Limitations

Currently, the experimental results are obtained only using two tasks: prefix-sum and mazes. Testing the validity of our findings on a more diverse set of tasks would increase our confidence in the fact that the phenomenon reported in this paper is universal. Moreover, it is not clear from our analysis which tasks would benefit from more test-time compute. While the necessity of additional test-time compute is obvious for prefix sum and mazes, it's less clear for perception-like tasks like image classification. Furthermore, we believe that a more systematic study of how the choice of solver (i.e. Anderson (first order) vs. Broyden (second order) solvers) impacts the empirical AA scores would strengthen our findings. We observe, for example, that networks trained with Broyden struggle to find fixed points at test time when run with fixed-point iterations. Expanding on these findings would improve our analysis and let us make more concrete suggestions to practitioners aiming to use the path-independence related concepts from this paper.

D Path Independence vs. Accuracy Plots Different Difficulty Levels

The path-independence vs. accuracy plots in Figure 4b uses averaged values over different problem difficulties. In Figure 7, we show how these correlations look like at different lengths on the prefix

sum task. The same figure also shows how the accuracy differs if one does inference using the training time forward pass budget. The main takeaways from this plot are:

- Out-of-distribution problem lengths (right top of the Figure) do indeed require larger inference depths - all models perform very poorly when inference is run with training-time forward pass budget.
- The correlation between path independence and accuracy - as the definition implies - is more apparent in the very large forward pass limit. In this regime, the correlation persists for all lengths.
- While networks with low PI values using the training-time budget occasionally perform worse when the test-time compute is increased, that doesn't happen with path-independent networks.

E Intervention Results on Different Difficulty Levels

Mirroring Section D, we provide how the accuracy correlates with path independence on 1) different problem difficulties (32 is in-distribution) and different forward pass budgets in Figure 8.

We especially emphasize the *mixed initialization* results (mixed initialization is one of the interventions that promotes path independence). Note that some of these networks (half of which are as shallow as 6 layers) do poorly in and out-of-distribution when the forward pass uses the training forward pass budget. Unsurprisingly, the AA scores for these networks are low in this condition. However, when additional test-time compute is provided, these networks reach perfect in-distribution accuracy, and near-perfect OOD accuracy (especially on length = 64 split). This demonstrates that training time convergence is not required for path independence. We find it surprising that with the help of a path-independence-promoting regularization procedure, it's possible to train very shallow (i.e. less than 6 layers) networks such that they can exploit additional test-time compute to achieve very high accuracy.

F Per-Instance Path Independence Analyses - Convergence vs. Path Independence

In addition to the analysis in Section 8, we also analyzed how test-time convergence correlates with path independence on a per-instance basis. The per-instance AA score vs. convergence (measured in terms of the L2 distance between the last two root solver iterates) can be seen in Figure 9. We used the 12 networks trained using the mixed initialization strategy (since these networks yielded the best in and OOD performance), and used 300 samples from the length splits 16, 32, 64, 128 and 256.

This analysis reaffirms our past observation that test-time convergence is not needed for path independence. The plot contains data from three different regimes:

- **Full convergence to a fixed point:** These datapoints, observed on the right bottom part of Figure 9, corresponds to samples where the solver nearly converges to a fixed point. This is almost always associated with high FPA scores, and good per-instance accuracy.
- **Limit cycles:** These datapoints, corresponding to the right-top part of Figure 9, correspond to cases where the solver doesn't converge to a fixed points, but enters an orbit around it. AA scores, as well as the per-instance accuracy values remain high in this regime as well, indicating that test-time convergence (to a fixed point) is not necessary for path independence, and good accuracy. Note that the boundary between the full-convergence regime and the limit cycle regime is gradual.
- **Divergence:** On a number of the samples, the solver diverges. These are associated with high residuals, low AA scores and low per-instance accuracies. This shows that the main source of per-instance lack of path independence on networks that are otherwise overwhelmingly path independent (on other samples) is almost always solver divergence.

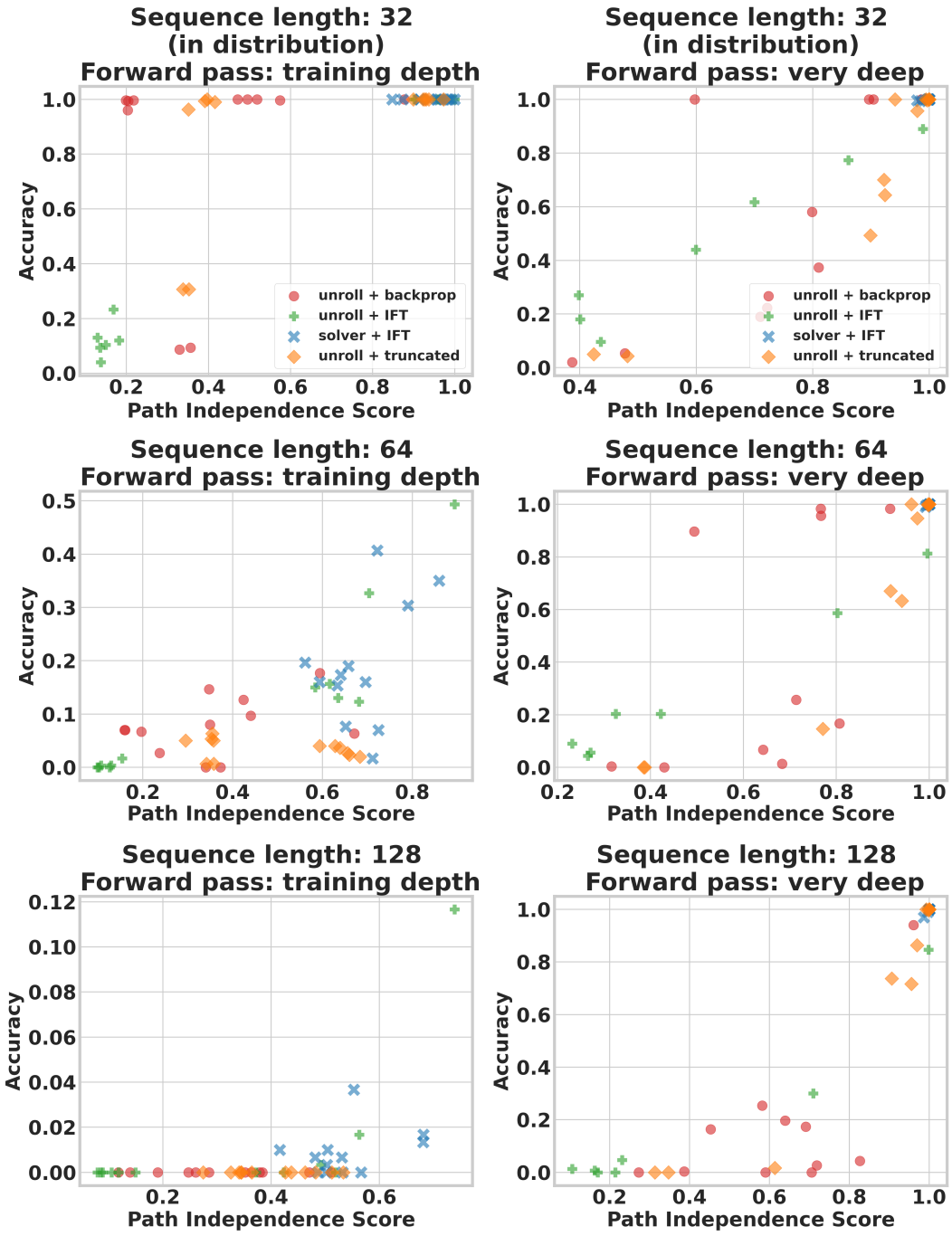

Figure 7: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in the large inference-time budget limit. Each row corresponds to a different problem difficulty, with the first row (length = 32) corresponding to in-distribution data.

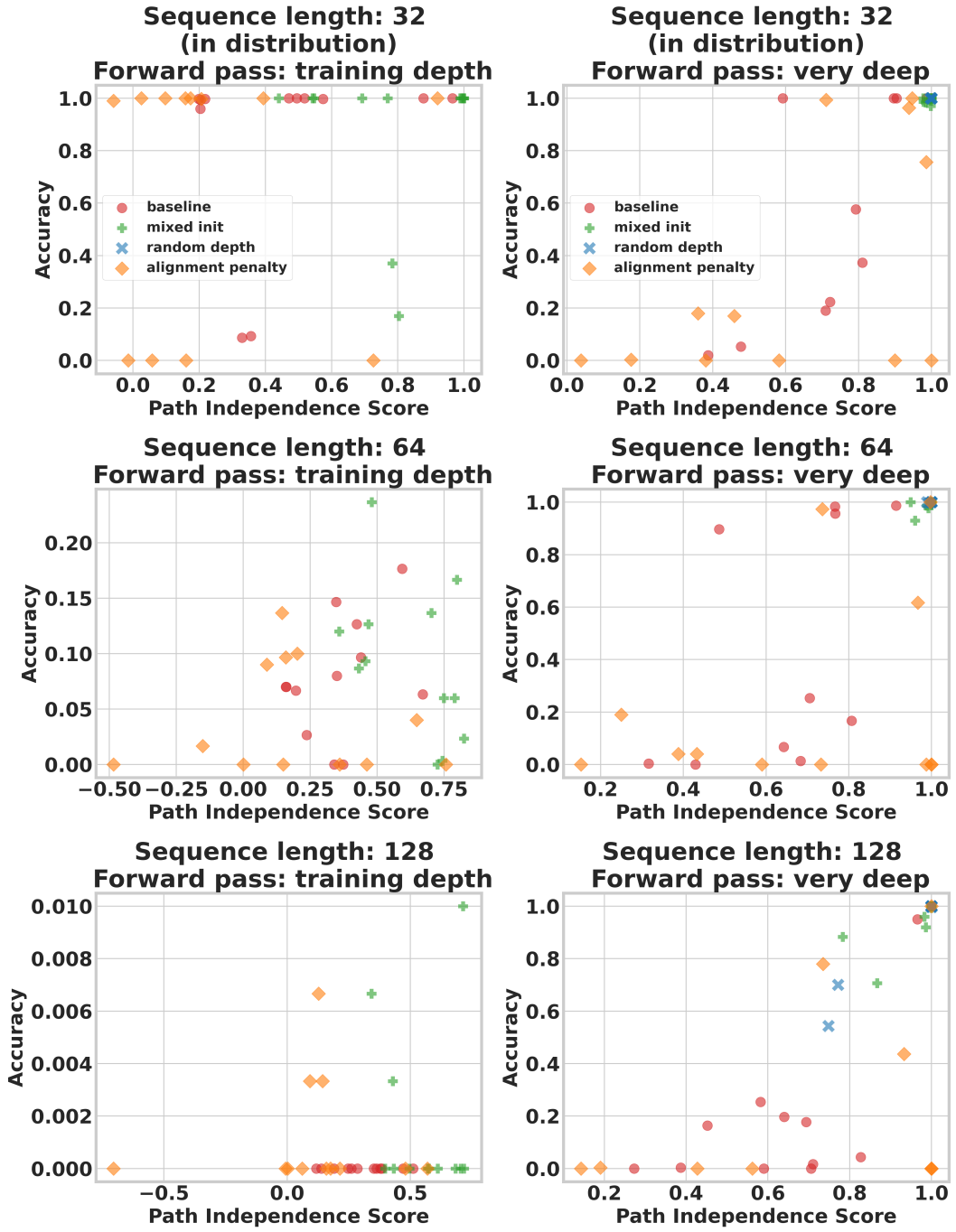

Figure 8: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in the large inference-time budget limit. Each row corresponds to a different problem difficulty, with the first row (length = 32) corresponding to in-distribution data.

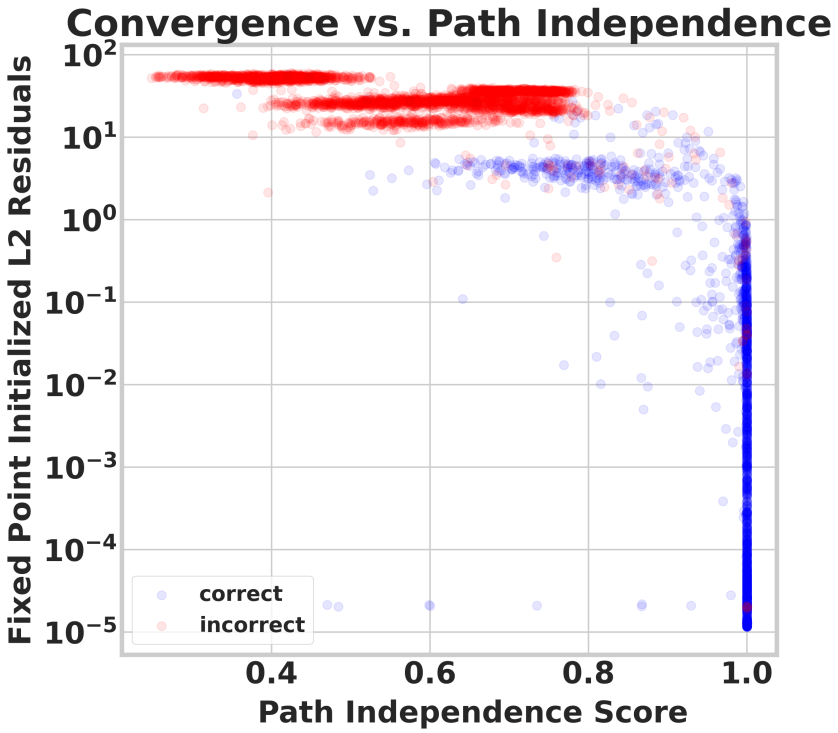

Figure 9: **Per-Instance Test-time Convergence vs. Path Independence**: The per-instance AA score vs. convergence (measured in terms of the L2 distance between the last two root solver iterates). Each example is labelled based on whether the network's prediction on it was correct (blue) or not (red). We used the 12 networks trained using the mixed initialization strategy (since these networks yielded the best in and OOD performance), and used 300 samples from the length splits 16, 32, 64, 128 and 256. While good convergence is almost always associated with high AA score values, the converse is not necessarily true: there are many samples on which convergence is poor, yet the AA score is high.

G Test Time Convergence and Path independence

We provide a per-instance analysis of test-time convergence in Figure 10. We display plots for per-instance residuals i.e. $\|f(x,z) - z\|_2$ for Broyden's method and naive fixed point iterations over solver steps. In addition, we also plot L2 norm between the fixed points of these solvers at every step. We provide plots for both in-distribution data i.e. 9×9 mazes and on more difficult problem instances i.e. 13×13 and 25×25 mazes. Across all the mazes, we observe that there are problem instances where both the solvers converge to the same limiting behavior (as indicated by low values of residuals and L2 norms). However, there are a considerable number of points where naive fixed point iterations converge to a limit cycle but still output correct predictions. This behavior can be seen on both in-distribution data and on harder problem instances. We find that the absolute residuals between the points in the limit cycles are a small percentage of the Euclidean norms of the points: 0.9% for 9x9 mazes, 0.49% for 13x13 mazes and 1.5% for 25x25 mazes which indicates that these limit cycles are localized. These examples of problem instances where both the solvers converge to different limiting behavior reaffirm our observation that convergence to the same fixed point at test-time is not a necessary condition for path independence.

H Results on the Blurry MNIST Task

We tested our path-independence hypothesis on task we call BlurryMNIST [Liang et al.]. This task involves training an image classifier on MNIST digits corrupted with small degrees of Gaussian blur, and testing the performance on significantly more corrupted ones. The blur corruption is implemented by convolving each image with Gaussian filters of differing standard deviations. We used standard

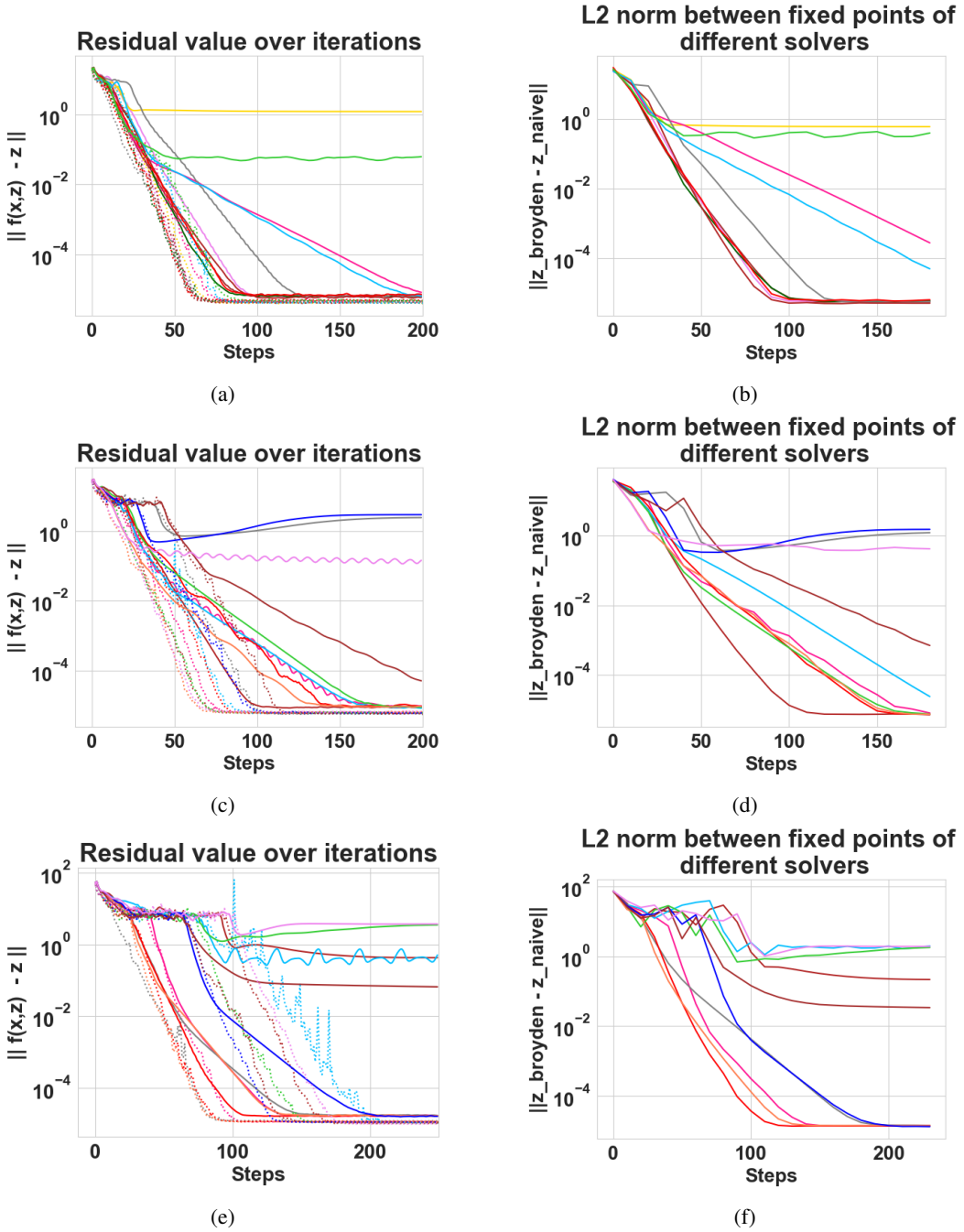

Figure 10: Different solvers display differing asymptotic behaviour but still achieve good upwards generalization (**Left column**) We display plots for the values of $||f(x,z) - z||_2$ over multiple solver steps for Broyden's method (dotted lines) and naive fixed point iterations (solid lines). (**Right column**)We also plot L2 norm between the fixed points obtained through fixed point solver and Broyden's method. Each line indicates one problem instance and for a given row, lines with same color are the same problem instance. The network was trained on 9×9 mazes, has an adversarial FPA score of 0.99, and achieves accuracy of 100% with both the solvers on all the displayed problem instances of mazes; **(first row)** 9×9 mazes i.e. in-distribution, **(second row)** 13×13 mazes. **(third row)** 25×25 mazes

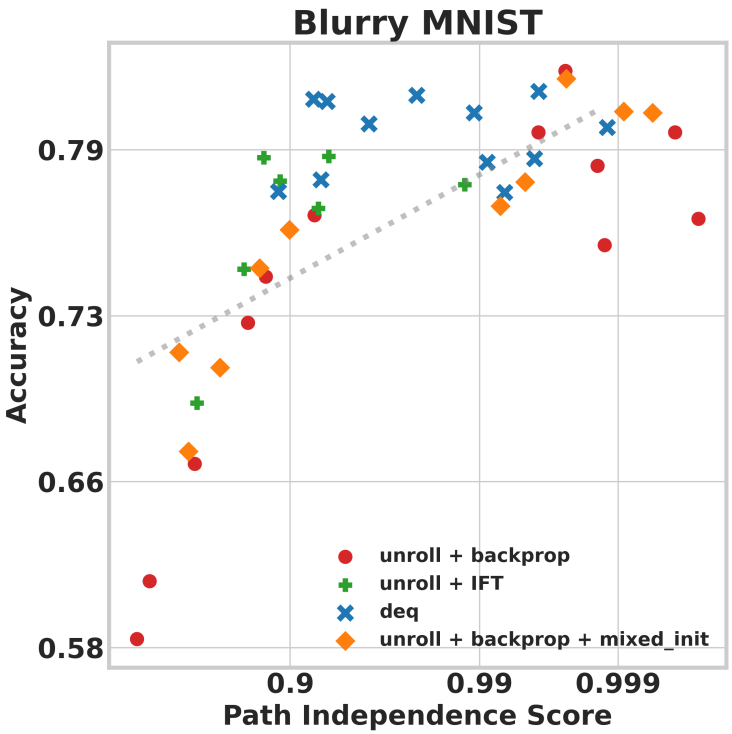

Figure 11: **BlurryMNIST results:** Path independence and generalization correlate on the BlurryMNIST dataset. This task involves training an image classifier on MNIST digits corrupted with small degrees of Gaussian blur, and testing the performance on significantly more corrupted ones.

deviations from 2 to 5.5 (in increments of 0.5) to generate each split. This dataset allows for testing in and out-of-distribution performance by way of training on a subset of the splits (i.e. the four lowest blur splits) and testing performance on all splits.

We trained a number of fully connected equilibrium models on the BlurryMNIST task and inspected whether path independence still correlates with accuracy. The results can be found in Figure 11. The correlation we reported in the main body of the paper also holds in the BlurryMNIST dataset. Note that the in-distribution error rates of the trained models vary between 1 and 5 percent.

I Results on Matrix Inversion Task

We also tested the connection between path independence and generalization on the Matrix Inversion task proposed by Du et al. [2022]. This task is concerned with learning to invert 20x20 matrices. Success is defined by how well the trained model works on matrices with worse condition numbers than those observed during training. Note that this task is qualitatively very different from all the others we considered before. We have summarized the results in Figure 12.

Using a fully-connected ResNet block as the equilibrium model cell of width 512, we trained a number of equilibrium models where we varied 1) the forward pass (fixed point iterations vs. solver) 2) the backwards pass (backprop gradients or implicit gradients) 3) learning rate (0.001, 0.0001 and 0.00001), 4) learning rate schedule (step decays of magnitude 0.5 at different points during training) and 5) whether layer normalization [Ba et al., 2016] was used or not. The results can be seen in Figure 12. We see a similar pattern that we saw on earlier tasks: lack of path independence correlates with poor generalization. Note that our best model matches the performance of the energy based model approach proposed by Du et al. [2022] in the matrix inversion task and significantly beats their baselines.

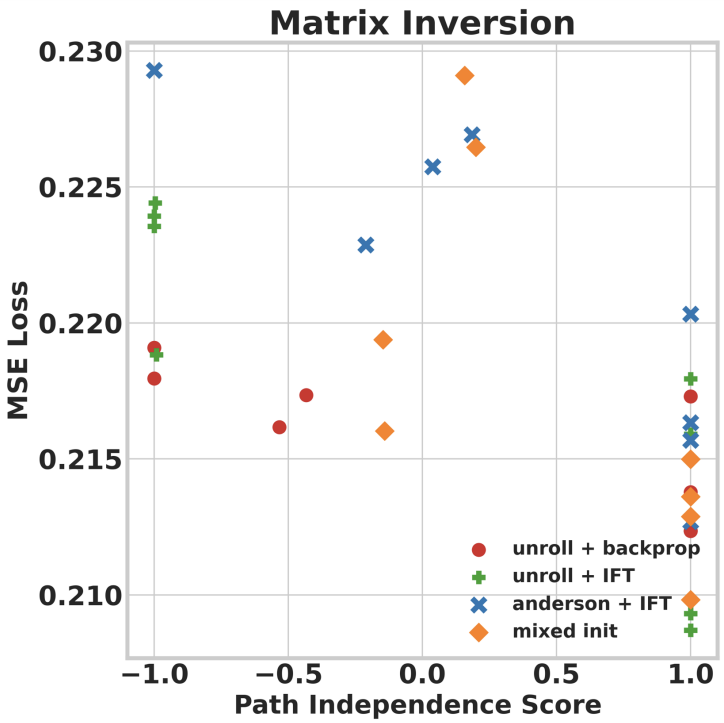

Figure 12: **Matrix inversion results:** We also tested the connection between path independence and generalization on the Matrix Inversion task proposed by Du et al. [2022]. Lack of path independence correlates with poor generalization in this task as well (note that **lower is better** in this task).

J Results on the Edge Copy Task

We also test our path independence hypothesis on tasks that take in a graph as an input. We consider the following the edge copy task proposed by Du et al. [2022], where the goal is to learn to simply output the input edge features, in a way that generalizes to larger graph sizes.

We used an equilibrium model cell that's compatible with graph tasks in the Edge Copy experiments, whose structure is shown in Figure 14. This cell is especially suited for edge regression tasks, since each application of the cell refines the edge features.

We trained a number of equilibrium models where we varied 1) the forward pass (fixed point iterations vs. solver) 2) the backwards pass (backprop gradients or implicit gradients) 3) learning rate (0.0001, , 0.000333 and 0.0001) and 4) whether layer normalization [Ba et al., 2016] was used or not. Each hyperparameter configuration was run twice with different seeds. The results can be seen in Figure 13. We see a similar pattern that we saw on earlier tasks: lack of path independence correlates with poor generalization. Note that our best equilibrium model outperforms the the energy based model approach proposed by Du et al. [2022] in this task.

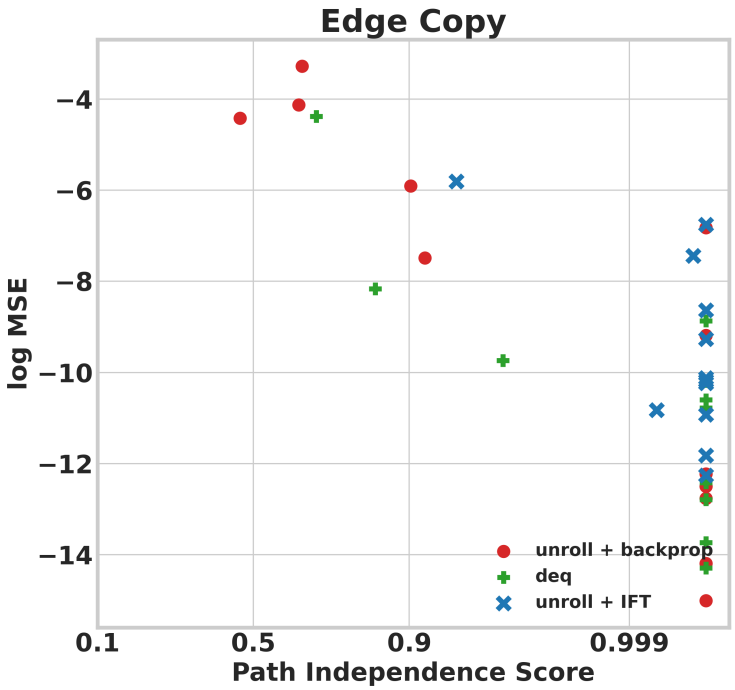

Figure 13: **Edge copy task:** The edge copy task requires learning to simply output the input edge features, in a way that generalizes to larger graph sizes. Lack of path independence correlates with poor generalization in the edge copy task as well. Note that **lower in better** in this task.

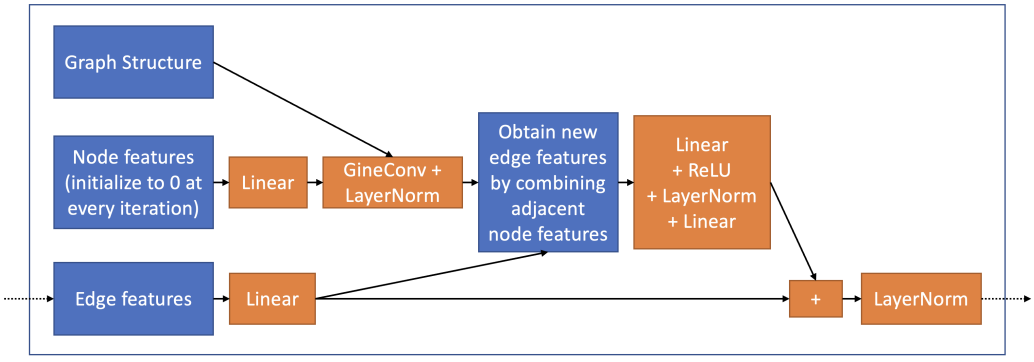

Figure 14: **A graph-processing equilibrium model cell:** In the edge-copy experiments, we used the equilibrium model cell illustrated above. The workhorse of this cell is the GINEConv operation [Hu et al., 2019], which fuses node and edge features to produce updated node features. This cell is especially suited for edge regression tasks, since each application of the cell refines the edge features.