Path Independent Equilibrium Networks Can Better Exploit Test-Time Computation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Designing networks capable of attaining better performance with an increased inference budget is important to facilitate generalization to harder problem instances. Recent efforts have shown promising results in this direction by making use of depth-wise recurrent networks. In this work, we reproduce the performance of the prior art using a broader class of architectures called equilibrium models, and find that stronger generalization performance on harder examples (which require more iterations of inference to get correct) strongly correlates with the *path independence* of the system—its ability to converge to the same attractor (or limit cycle) regardless of initialization, given enough computation. Experimental interventions made to promote path independence result in improved generalization on harder (and thus more compute-hungry) problem instances, while those that penalize it degrade this ability. Path independence analyses are also useful on a per-example basis: for equilibrium models that have good in-distribution performance, path independence on out-of-distribution samples strongly correlates with accuracy. Thus, considering equilibrium models and path independence jointly leads to a valuable new viewpoint under which we can study the generalization performance of these networks on hard problem instances.

1 Introduction

One of the main challenges limiting practical applicability of modern deep learning systems is the ability to generalize outside the training distribution [Koh et al., 2021]. One particularly important type of out-of-distribution (OOD) generalization is *upwards generalization*, or the ability to generalize to more difficult problem instances than those encountered at training time [Selsam et al., 2018, Bansal et al., 2022, Schwarzschild et al., 2021b, Nye et al., 2021]. Often, good performance on more difficult instances will require a larger amount of test-time computation, so a natural question arises: how can we design neural net architectures which can reliably exploit additional test-time computation to achieve better accuracy?

We argue that a key determiner of whether a learned model can exploit additional test-time computation is whether the dynamical system corresponding to the model is *globally stable*; that is, whether the learned model converges to the same limit point (or limit cycle), regardless of the initialization of the system or the number of times the components of the model are applied. For example, a simple integrator $x_{t+1} = x_t + 1$ is clearly *not* globally stable, as it the final state depends on the initial state and the number of iterations run; conversely, the system $x_{t+1} = (x_t + 1)/2$ *is* globally stable, as it will converge to the solution $x_T = 1$ as $T \to \infty$ regardless of the initial condition of x_0. However, the term "global stability" is slightly overloaded, as it sometimes requires convergence to a single point [Slotine et al., 1991], and sometimes just implies the system is convergent everywhere, even if to different points[Wang et al., 2003, Sriperumbudur and Lanckriet, 2009]. We thus choose the term

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

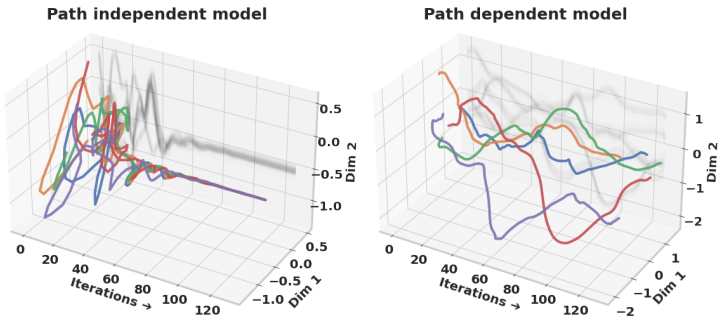


Figure 1: Trajectories of path independent models converge to the same hidden state for a given input, regardless of initialization, whereas the trajectories of path dependent models depend on initialization. Here, we display five trajectories with different initializations obtained from a path independent (left) and path dependent model (right) on the prefix-sum task, projected onto two random directions.

path independence to refer specifically to the fact that the system will converge to the same limiting behavior (whatever that might be) regardless of the initial state of the system.

Intuitively, path independent algorithms can more easily take advantage of additional test-time iterations than path dependent ones. For instance, gradient descent applied to a convex objective is path independent, and correspondingly when confronted with a more ill-conditioned problem instance, one can compensate by increasing the number of iterations. Conversely, a weather simulation is path dependent, and extending the simulation won't yield more accurate predictions of a given day's weather. Based on this intuition, we hypothesize that path independence of a learned model is a key determiner of whether it can take advantage of an increased test-time iteration budget.

In principle, neural nets have the capacity to represent many different approaches to solving a problem; for instance, even vanilla RNNs are Turing-complete [Siegelmann and Sontag, 1995]. However, any particular combination of architecture and training regime may exhibit a strong inductive bias towards a particular approach. For instance, deep equilibrium models [Bai et al., 2019] use implicit weight updates which depend only on the final hidden state (not the path), and hence ought to encourage path independence, despite the ability to represent path-dependent functions. We hypothesize that the degree of path independence has a fundamental impact on generalization:

> *Path Independence Hypothesis:* Models which successfully fit the training distribution with a path independent function are better able to exploit more test-time iterations to achieve higher accuracy, compared to those which fit the training distribution with a path-dependent function.

We first introduce a metric for measuring the path independence of a learned model, the *Fixed Point Alignment (FPA)* score. On two algorithmic generalization benchmarks introduced by Schwarzschild et al. [2021a,b], we show that a model's FPA score is strongly correlated with its accuracy when far more iterations are used at test time than at training time. In general, we find that path independent models increase their accuracy monotonically with the number of test-time iterations, whereas path dependent models degrade when the number of test time iterations exceeds those at training time. We find that previously introduced architectural features which improve generalization, including input injection and implicit updates which are explicitly captured by equilibrium fixed point models, improve both the path independence and the accuracy. Furthermore, we perform an experimental manipulation by introducing regularizers which directly promote or punish path independence. We find that these manipulations, while increasing or decreasing the FPA score, also have the corresponding effect on accuracy.[1]

2 Background

Equilibrium Models Equilibrium models treat computing internal representations as a fixed-point finding problem. [McClelland and Rumelhart, 1989, Liao et al., 2018, Bai et al., 2019] Concretely, letting $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and $f_{\boldsymbol{w}} : \mathbb{R}^{n_x \times n_z} \mapsto \mathbb{R}^{n_z}$ stand for an input and the equilibrium model function (or

[1] Code will be released along with the paper

"cell") parametrized by $\boldsymbol{w} \in \mathbb{R}^{n_w}$ respectively, equilibrium models aim to solve for the fixed point $\boldsymbol{z}^* \in \mathbb{R}^{n_z}$ that satisfies $\boldsymbol{z}^* = f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}^*)$. The cell $f_{\boldsymbol{w}}$ might represent anything from a fully connected layer to a transformer block [Vaswani et al., 2017]. We emphasize that $f_{\boldsymbol{w}}$ directly depends on the input $\boldsymbol{x}$ – following existing literature, we refer to this as *input injection*. The outcome of the fixed point finding process might depend on the initial fixed point guess. To make this dependency explicit, we define the function $\text{FIX} : \mathbb{R}^{n_x \times n_z} \mapsto \mathbb{R}^{n_z}$ that maps an input $\boldsymbol{x}$ and an initial guess for the fixed point $\boldsymbol{z}_0$ to an output that satisfies the fixed point equation $\text{FIX}_{f_w}(\boldsymbol{x}, \boldsymbol{z}_0) := f_{\boldsymbol{w}}(\boldsymbol{x}, \text{FIX}_{f_w}(\boldsymbol{x}, \boldsymbol{z}_0))$. The behaviour of FIX depends on the solver that's used to find fixed points.

The most straightforward approach to solve for fixed-points is the *fixed point iteration* method, which recursively applies the function $f_{\boldsymbol{w}}$ on the internal representations $\boldsymbol{z}$ (i.e. $\boldsymbol{z}_{t+1} = f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}_t)$). If certain conditions are satisfied, (such as the fixed iterations being globally contractive – more general conditions are discussed below), this procedure converges [2] to a fixed point: $\boldsymbol{z}^* = f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}^*)$. As solving for fixed points exactly is expensive, fixed point iterations are often terminated after a fixed number of steps or when the norm of the difference between subsequent iterates falls below a pre-selected threshold. The model weights can be updated using gradients computed via backpropagating through the full forward computational graph.

If one commits to using fixed point iterations as the root solver, then the output $\boldsymbol{z}^*$ of equilibrium models can be interpreted as the infinite-depth limit of an input-injected, weight-tied model $f^{\infty}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}^*) = \lim_{n\to\infty} f^{(n)}_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}_0)$ where the notation $f^{(n)}$ stands for n repeated applications of f on its own output, and $\boldsymbol{z}_0$ stands for the fixed point initialization.

Deep Equilibrium (DEQ) Models Deep equilibrium models [Bai et al., 2019] solve for fixed points explicitly using black-box root finders, such as Broyden's method [Broyden, 1965] or Anderson acceleration [Anderson, 1965]. In order to avoid explicitly differentiating through the root-finding procedure, DEQ models utilize implicit differentiation to compute gradients. Concretely, letting $g_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}) = f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}) - \boldsymbol{z}$ for a fixed point $\boldsymbol{z}^*$, the Jacobian of $\boldsymbol{z}^*$ with respect to the DEQ model weights can be given by:

$$\frac{d\boldsymbol{z}^*}{d\boldsymbol{w}} = -\left(\frac{\partial g_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}^*)}{\partial \boldsymbol{z}^*}\right)^{-1} \frac{\partial f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}^*)}{\partial \boldsymbol{w}} \tag{1}$$

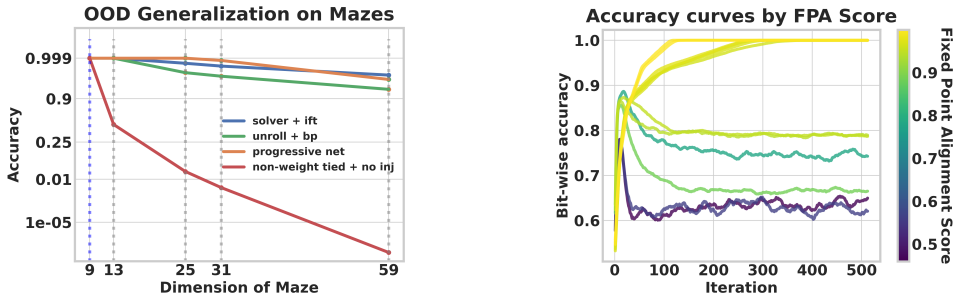
Computing the inverse of a Jacobian can become computationally expensive. Recent works [Geng et al., 2021a, Fung et al., 2021] have shown that the inverse-Jacobian term in Eq. (1) can be replaced with an identity matrix i.e. Jacobian-free or an approximate inverse-Jacobian [Geng et al., 2021b] without affecting the final performance of DEQs. This approximation makes the backward pass of DEQs inexpensive and lightweight. DEQ models have been shown to achieve state-of-the-art performance on various tasks including language modelling [Bai et al., 2019], image recognition, semantic segmentation [Bai et al., 2020], object detection [Wang et al., 2020], and graph modeling [Gu et al., 2020, Liu et al., 2021a].

DEQs vs. Depthwise Recurrent Models Both DEQs and depthwise recurrent (i.e. weight-tied, fixed-depth) networks leverage weight-tying *i.e.,* they apply the same transformation at each layer, $f^{[i]}_{\omega} = f_{\omega}\ \forall i$. However, the key difference between the two is that weight-tied fixed depth networks require backpropagation through an explicit stack of layers but DEQs directly solve for fixed points using black-box solvers during the forward pass and use implicit differentiation to compute gradients.

Convergence As alluded above, in order to guarantee convergence to a unique fixed point, it suffices for the cell of the equilibrium model to be *contractive* over its input domain [3] (i.e. the singular values of its Jacobian all lie below 1). Previous work have leveraged Lipschitz constrained cells to ensure contractivity[Revay et al., 2020]. Other approaches for ensuring global convergence exist: the monotone DEQ architecture guarantees global convergence by utilizing a DEQ parametrization that bears similarities to solutions to a particular form of monotone operator splitting problem [Winston and Kolter, 2020]. Unrestricted DEQs aren't constrained enough to guarantee convergence: they can easily express globally divergent vector fields that prohibit the existence of fixed points. It is, therefore, interesting that they can and often learn path independent solutions. Also note that the "infinite-depth weight tied network" interpretation of DEQs is less general than the implicit formulation presented above, as the latter admits unstable fixed points as well.

[2] Divergence is also a possible outcome, rendering the output of equilibrium models unusable.

[3] This is known as Banach Fixed Point Theorem.



(a) (b)

Figure 2: (**left**) Strong upward generalization on mazes by PI models. Models were trained on 9×9 sized mazes and tested for upward generalization on larger mazes. y-axis uses probit transformation. (**right**) PI models are better able to make use of additional test-time computation. We trained models with varying number of training-time iterations, learning rate and weight norm application. Bit-wise accuracies are evaluated and averaged over different string-lengths.

3 Upwards Generalization with Equilibrium Models

To study the effects of test time computation, it is useful to consider tasks with an explicit difficulty parameter, so that the learned models can be tested on more difficult instances which require a large number of iterations to solve correctly. We focus in particular on two algorithmic generalization benchmarks introduced by Schwarzschild et al. [2021a,b]: **prefix sum** and **mazes**. In this section, we first replicate their findings on these problems, and then extend them to a wider variety of architectures considered in this paper, most notably deep equilibrium models.

Tasks Prefix-sum is a sequence-to-sequence task whereby the network is given a sequence of 0-1 bits, and is trained to output, for each bit, the parity of all of the bits received since the beginning of the sequence until the current bit. We train on 10,000 unique 32-bit binary strings, and report results on binary strings of other lengths. The mazes task is also an image-to-image task, where the input is a three-channel RGB image. The 'start' and 'finish' positions are marked by a red and a green square respectively; walls are marked in black. The output is the optimal path in the maze that connects these two points without passing through the walls. We train on 50,000 small mazes of size 9×9, and report upward generalization results on larger mazes. Instances of each of these problems are provided in the Supplementary Material.

Strong Upward Generalization Figure 2a shows that equilibrium models demonstrate very strong upward generalization performance compared to non weight-tied fixed-depth models. Moreover, Figure 2b shows that increasing inference depth consistently improves performance—especially on harder problem instances.

Terminology and Abbreviations We use the term 'solver' to refer to the use of black-box root finders like Anderson acceleration to find fixed points of a DEQ. These networks use implicit gradients (IFT) for the backward pass. The term 'unroll' refers to equilibrium models that use regular fixed-point iterations to compute the equilibrium point. We use the abbreviation 'bp' to refer to backpropagation gradients, and 'inj' to refer to input injection. The term 'progressive net' refers to the deep thinking networks trained with progressive training as proposed by Bansal et al. [2022]. We use 'PI' and 'non-PI' to refer to path independent and non dependent networks, respectively.

4 Path Independence

Having intuitively motivated the idea of path independence in Sec. 1, we now formulate a precise definition: we say that the computation performed by a recurrent operator computing function $f_{\boldsymbol{w}}$ on an input $\boldsymbol{x}$ is path independent if it converges to the same limiting behavior regardless of the current state $\boldsymbol{z}_t$. As a special case, if the computation is convergent, this property is equivalent to the existence of a unique fixed point $\boldsymbol{z}^*$ such that $f_{\boldsymbol{w}}^{\infty}(\boldsymbol{x}, \boldsymbol{z}_0) = \boldsymbol{z}^*$ for any $\boldsymbol{z}_0$. However, our definition allows for other behaviors such as limit cycles (see Section 7).

Some architectures guarantee the path independence property (see Section 2). However, most common DEQ architectures—and the ones we will be using throughout this paper—have the expressive power

to easily learn multiple fixed points per input. Since it is unclear whether architectures enforcing the contraction property lose expressiveness [Bai et al., 2021], we focus our investigation on architectures not guaranteed to be path independent.

PI networks represent a different model for computation than standard feed-forward networks: instead of learning an entire computational graph to map inputs to outputs, they only have to *learn where to stop*. We dedicate the rest of the paper on exploring the *Path Independence Hypothesis*—the idea that path dependent networks can better exploit test-time computation and hence show strong upward generalization from easy to hard problem instances of algorithmic tasks.

Before establishing a connection between path independence and out-of-distribution generalization, we first describe two architectural components that are *necessary conditions* for achieving path independence, then proceed to describe a metric to quantify degrees of path independence.

4.1 Architectural Components Necessary for Path Independence

Past work has observed that *weight tying* and *input injection* are both crucial for upwards generalization [Bansal et al., 2022]. We observe that both architectural components are also necessary for a learned model to be PI [4]. Without weight tying, the network is constrained to have a fixed forward depth, so it is meaningless to talk about the limiting behavior in large depth. Input injection ensures that the equilibrium point depends on the input despite having an "infinite depth". Without input injection, a PI network would necessarily forget the input; hence, any model which successfully fits the training distribution must be path dependent.

Interestingly, both architectural motifs are also key components of deep equilibrium models [Bai et al., 2019]; in that work, the motivation was to enable efficient gradient estimation via the implicit function theorem (IFT)— a concept closely related to path independence, since the premise of the IFT gradient estimator is that only the final hidden state matters, not the path taken to get there. It is striking that two separate lines of work would converge on the same architectural motifs, one motivated by generalization and the other by a variant of path independence. We believe this is not a coincidence.

Reproducing the results of Bansal et al. [2022], in Figure 2a we show upward generalization performance using both equilibrium models and progressive nets [Bansal et al., 2022] – and the lack thereof using non-input injected networks. For the remainder of this paper, we focus on architectures with both input injection and weight tying.

4.2 Quantifying Path Independence

We propose a simple metric to quantify path independence based on the directional alignment of the fixed points computed with the same input, but different initializations. We name this metric the *Fixed Point Alignment (FPA) score*. Pseudocode to compute the metric is given in Algorithm Box 1. The FPA score is the average cosine similarity between the fixed point obtained with the training time initialization (often simply the zero vector) and the fixed points obtained when one initializes the solver *using the fixed points computed on different inputs*. Higher FPA scores (with 1 being the highest value) imply higher degrees of path independence. In Section 5, we show a strong correlation between path independence and accuracy using the FPA score.

FPA score is cheap to compute, is a reliable indicator of path independence (see below) and is unitless, meaning networks obtained from different training runs can be compared on equal footing. See SM for other metrics we've considered for quantifying path independence and why we settled on FPA score.

Stress-testing the FPA score To stress-test the extent to which the FPA score really measures path independence, we search for *adversarial initializations* that are optimized to result in distinct fixed points, hence low FPA values. (Unlike adversarial examples, this attack is not constrained to an ε-ball.) We use the L-BFGS [Liu and Nocedal, 1989] optimizer, and repeat the search multiple times starting from different fixed point initializations. We include pseudocode in the Supplementary material.

The results of the adversarial stress test can be seen in Table 1. The results corroborate that the FPA score is indeed a reliable measure of path independence; while it is not possible to find adversarial initializations for high FPA networks (indicating high path independence), low FPA networks can easily be adversarially initialized to be steered away from the original fixed point estimate.

[4] Our definition also admits non-input injected models to be path independent if they're representing constant functions (i.e. input independent). We don't consider such cases in our analyses.

Algorithm 1 Fixed Point Alignment Score

Input: A batched input $\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}$, an operator $f_{\boldsymbol{w}}$

Initialize: $\begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \mathbf{0}$

Define: $h(\boldsymbol{y}_1, \boldsymbol{y}_2) = \frac{\boldsymbol{y}_1}{\|\boldsymbol{y}_1\|_2} \cdot \frac{\boldsymbol{y}_2}{\|\boldsymbol{y}_2\|_2}$

Compute $\begin{bmatrix} \boldsymbol{z}_1' \\ \boldsymbol{z}_2' \end{bmatrix} = \mathrm{FIX}_{f_{\boldsymbol{w}}}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix}\right)$

Interchange and reinitialize fixed points

Compute $\begin{bmatrix} \boldsymbol{z}_1'' \\ \boldsymbol{z}_2'' \end{bmatrix} = \mathrm{FIX}_{f_{\boldsymbol{w}}}\left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{z}_2' \\ \boldsymbol{z}_1' \end{bmatrix}\right)$

return $\mathrm{average}(h(\boldsymbol{z}_1'', \boldsymbol{z}_1'), h(\boldsymbol{z}_2'', \boldsymbol{z}_2'))$

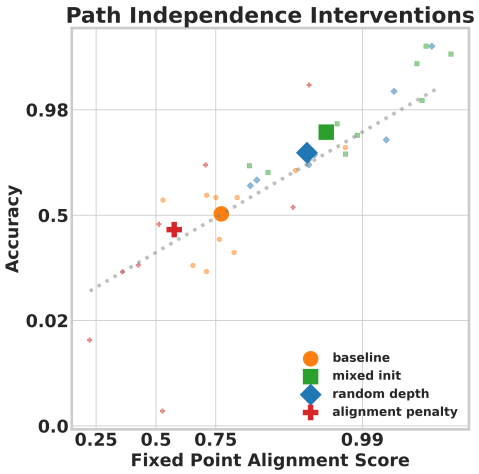


Figure 3: **(left) Fixed Point Alignment Score Algorithm:** We provide the algorithm for a simple illustrative case of two inputs. In practice, we consider larger batches. **(right) Promoting path independence improves generalization in the prefix sum task:** Interventions that are designed to promote path independence (initializing fixed points with random noise or running the fixed point solver with stochastic budget) improves generalization. Conversely, those that hurt path independence (penalty term that that directly penalizes fixed point alignment) leads to poorer generalization.

Model	Task	FPA ↑	Accuracy (%)	Attacked FPA ↑	Attacked Acc. (%)
Non-PI network	Maze	0.32	87.12	0.09	0
PI network	Maze	1.00	100	1.00	100
Non-PI network	Prefix sum	0.62	66.66	0.18	0
PI network	Prefix sum	0.99	100	0.99	100

Table 1: FPA scores for PI vs non-PI networks computed on 13×13 mazes and 64 bit prefix sum. Attacked FPA refers to the cosine similarity between the fixed point from zero initialization and an adversarial initialization.

5 Path Independence Correlates with Upward Generalization

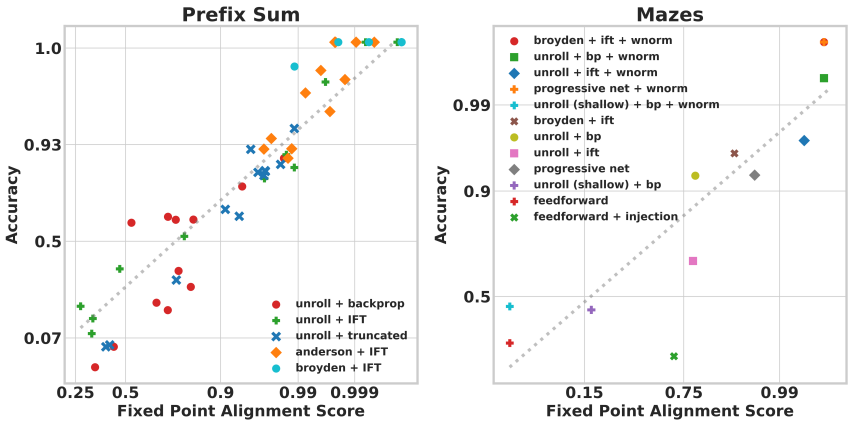
Is path independence (as measured by the FPA score) is a strong predictor of upwards generalization? We took the trained networks from Section 3, computed their average FPA Scores on in- and out-of-distribution splits and inspected whether the FPA scores are correlated with upward generalization.

On prefix sum experiments, we varied 1) network depth, 2) whether or not weight norm (wnorm) [Salimans and Kingma, 2016] was used or not [5] 3) learning rate (one of [0.01, 0.001, 0.0001]) 4) forward solver (fixed point iterations or Anderson acceleration [Anderson, 1965], and 5) the gradient estimator (backprop or implicit gradients)[6]. On the maze experiments, we varied 1) network depth, 2) use of weight norm 3) forward solver (fixed point iterations or Broyden solver [Broyden, 1965]), and 5) the gradient estimator (backprop or implicit gradients)

Figure 4 displays our findings. We evaluated performance on a mixture of in and OOD validation data - results on individual data splits can be found in the Appendix. The results show a strong correlation between FPA score and accuracy when the inference depth is large enough. This shows that PI networks that allow for scaling test-time compute improve test-time accuracy (also see Figure 2b). The in-distribution validation performance of non-PI networks degrade with deeper inference depths. Unsurprisingly, these networks generalize poorly on harder problem instances that provably require deeper inference depths.

[5] Bai et al. [2019] report that weight norm helps stabilize the training of DEQ models.

[6] Note that the deep equilibrium model (DEQ) setup [Bai et al., 2019] correspond to using a root solver (such as Anderson) for the forward pass and implicit gradients for the backward pass.



(a) Prefix Sum (b) Mazes

Figure 4: High FPA scores correlate with good upward generalization. For a given choice of an architecture and a task, the reported numbers are averaged over problem instances of different dimensions. We apply the probit transformation along both the axes, following Miller et al. [2021]. The accuracies and the FPA scores are capped at 0.999 for compatibility with the probit transform.

6 Experimental Manipulations of Path Independence

Previous sections demonstrated a strong correlation between path independence and the ability to exploit test-time iterations. Unfortunately, we can't make a causal claim based on those studies: the observed effect could have been due to an unobserved confounder. In this section, we intervene directly on path independence by imposing regularizers which directly encourage or penalize path independence. We find that interventions designed to *promote* path independence also improve generalization, while interventions designed to *reduce* path independence also hurt generalization.

6.1 Promoting Path Independence via Randomized Forward Passes

A straightforward way to encourage path independence is simply to initialize the hidden states with random noise during training. To this end, we experimented with initializing the hidden states with zeros on half of the examples in the batch, and with standard Gaussian noise on the rest of the examples. The reason to include the zero-initializations at training time is that we initialize from zeros at test time - not including this initialization during traning time causes a distribution shift.)

Another way to promote path independence is simply running the forward solver with randomized compute budgets/depths during training time. While a path independent solution can be expected to be robust against this intervention, a path dependent one will fail.

We took the training configurations of the 12 prefix-sum networks described in Section 5 that use fixed point iterations in their forward pass, and backpropagation gradient in their backward pass, and retrained them separately with the aforementioned mixed initialization and randomized depth strategies without modifying any other experimental conditions. As can be seen in the Figure 3, the interventions lead to strong test-time path independent neural networks, while also reliably improving in- and out-of-distribution validation accuracy. We especially emphasize that shallow networks trained with mixed initialization actually remain far from having high FPA scores using the training-time forward pass conditions due to lack of convergence. However, since the mixed initialization strategy results in path independent networks, scaling up test-time compute budget leads to high FPA scores, and therefore high upwards generalization.

6.2 Penalizing Path Independence via the Fixed Point Alignment Penalty

Does an intervention that results in less path independence also result in poorer upwards generalization? Like in the mixed initialization experiment, we retrained the 12 unroll + backpropagation networks with an additional auxiliary loss term that penalizes the dot product between the fixed points computed from the same input, but different initializations sampled from standard Gaussian noise. Figure 3 shows that this intervention succeeded in pushing the FPA scores down, while also keeping the accuracy on the same trend line.

Model	Residual		FPA score ↑		Accuracy (%) ↑	
	In-dist	OOD	In-dist	OOD	In-dist	OOD
DEQ (phantom grad.)	11.83	0.016	0.96	0.99	99.96	99.88
DEQ (IFT)	1.4	0.011	0.99	0.99	99.99	100

Table 2: Residual, FPA score, and Accuracy for DEQ trained with IFT vs in-exact gradients. OOD results have been computed on 25×25 mazes

7 Disambiguating Convergence and Path Independence

Is convergence necessary for path independence? We answer this statement in the negative, and show that neither training-time convergence nor test-time convergence is required for path independence. Instead convergence to the same limiting behavior regardless of initialization is important.

Training Time Convergence We consider two deep equilibrium models (DEQs) trained on the Mazes task—one trained with implicit gradients computed via implicit function theorem (IFT), and the other trained with phantom gradients [Geng et al., 2021b]. We report the values of residuals (*i.e.,* $\|f_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{z}) - \boldsymbol{z}\|_2$), FPA scores and accuracies observed for in- and out-of-distribution data for mazes in Table 2. We observe that DEQs trained with phantom gradients have higher values of in-distribution residuals but are path independent, as indicated by their high FPA scores, and show strong upward generalization as indicated by their good accuracy.

The mixed-initialization intervention described in Section 6.1 also leads to a separation between training-time convergence and path independence. We found that it is possible to train very shallow (*i.e.,* 5 layer) unrolled networks that, while being very far from converging during training and attaining poor in-distribution generalization, are able to achieve perfect performance when run for many more iterations during test time. Detailsare provided in the Supplementary material.

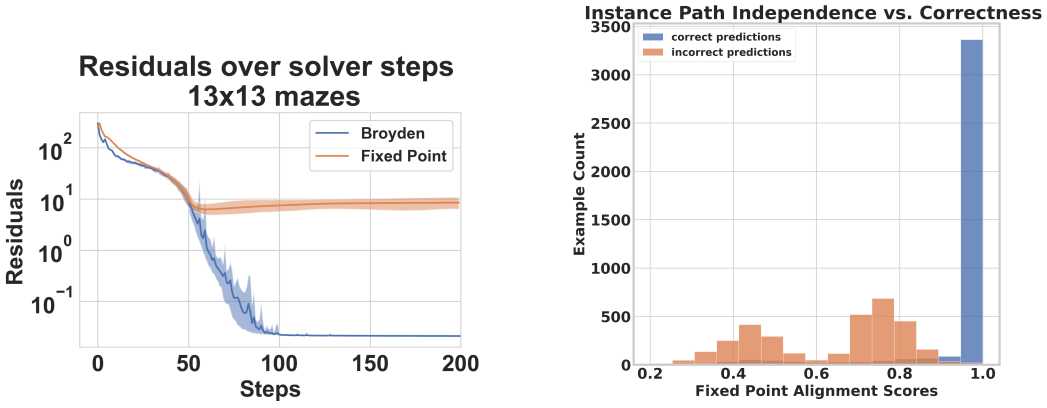
Test Time Convergence From Table 2, one might conclude that test time convergence is important for path independence. However, we show that this connection is not necessary, and convergence to the same fixed point is not a required condition for path independence. We study test time convergence properties of an unrolled weight-tied input-injected network trained with backpropagation under different solvers. This network is path independent, as indicated by its high FPA scores (0.99) on both in- and out-of-distribution data. We visualize the values of residuals observed at test time with fixed point iterations and Broyden's method in Figure 5a. Both these solvers converge to different limit cycles but still show good upward generalization.

8 Path Independence on a Per-Example Level

The connection between path independence and prediction correctness also largely holds on a per-instance basis. Using the prefix-sum networks trained with the mixed-initialization strategy (the most performant group of networks in our intervention experiments), we plotted the distribution of per-instance fixed point alignment scores, colored by whether the prediction on that instance was correct or not in Figure 5b. This suggests that path independence can be used as a valuable sanity-check to determine whether a prediction is correct or not without the need for any label data, both in- and out-of-distribution. We provide a more in-depth per-instance analysis in the Supplementary material.

9 Related Work

There is a long line of research on neural networks that can adapt their computational budget based on complexity of task they are learning to solve—akin to the intrinsic mechanism in humans to reason and solve problems. Schmidhuber [2012] introduced self-delimiting neural networks which are a type of recurrent neural networks (RNNs) that adapt their compute based on the output of a special "halt" neuron. Adaptive computation time (ACT) [Graves, 2016] also uses the output of a sigmoidal halting unit to determine the termination condition of RNN, but it avoids long "thinking" time by explicitly penalizing it. Subsequent works have successfully applied variants of ACT in image classification and object detection [Figurnov et al., 2017], visual reasoning [Eyzaguirre and Soto, 2020], Transformers [Vaswani et al., 2017] for language modelling [Dehghani et al., 2019, Elbayad et al., 2020a, Liu et al., 2021b], and recognizing textual entailment [Neumann et al., 2016]. PonderNet [Banino et al., 2021] reformulates the halting policy of ACT as a probabilistic model, and adds a regularization term in the loss objective to encourage exploration. With these additions, PonderNet can extrapolate to more



(a) (b)

Figure 5: (**Left**) Different solvers display differing asymptotic behaviours, but still achieve good upwards generalization. Here, the network has an adversarial FPA score of 0.99, and achieves accuracy of 99.98% (fixed point iterations), and 99.97% (Broyden solver) respectively on 13×13 mazes; (**Right**) Per-instance path independence is highly correlated with the correctness of predictions.

difficult examples on parity task. In this work, we do not optimize or penalize the network for the number of computational steps. Our main goal is to understand the underlying mechanism that results in scalable generalization of equilibrium models on harder problem instances.

Another family of models with the property of adaptive inference compute budget is early exit networks [Teerapittayanon et al., 2016, Laskaridis et al., 2021]. These networks have multiple additional "exit" prediction heads along their depth. At inference time, the result that satisfies an exit policy is selected as the prediction output. This approach of designing adaptive networks has been adapted both in natural language processing [Schwartz et al., 2020, Soldaini and Moschitti, 2020, Elbayad et al., 2020b, Zhou et al., 2020, Liu et al., 2020] and vision [Li et al., 2017, Wang et al., 2018, Xing et al., 2020, Kouris et al., 2021]. Most of these architectures have complex sub-modules that are trained in multiple stages, and require complex exit policies. In contrast, equilibrium models have a simple architecture, and can use root solvers to efficiently solve for the fixed point at inference.

More complex transformer-based language models like GPT-3 also struggle to generalize well on simple algorithmic tasks like addition [Brown et al., 2020]. Recent work by Nye et al. [2021] shows that transformers can be trained to perform well on algorithmic tasks and generalize on OOD data by emitting the intermediate steps of algorithm to a buffer called 'scratchpad'. Using a scratchpad enables the model to revisit its errors and correct them.

Our current work is closely related to previous work by Schwarzschild et al. [2021b] on easy to hard generalization of RNN on algorithmic tasks. A subsequent work [Bansal et al., 2022] proposes architectural choices and training mechanisms that enables weight tied networks to converge to a fixed point, and generalize on harder problem instances. We identify that these networks belong a more general class of networks called equilibrium models that learn solutions that are path independent, and can therefore generalize on harder problem instances.

10 Conclusion

Being able to attain better levels of performance using a larger inference-time compute budget is a feat that eludes most standard deep learning architectures. This is especially relevant for tasks that require *upwards generalization*, *i.e.,* the ability to generalize from easy problem instances to hard ones. We show that equilibrium models are capable of displaying upwards generalization by exploiting scalable test-time compute. We link this to a phenomenon we call *path independence*: the tendency of an equilibrium network to converge to the same limiting behavior given an input, regardless of the initial conditions. We investigate this phenomenon through careful experiments and verify that path independent networks indeed generalize well on harder problem instances by exploiting more test time compute. Moreover, interventions on training conditions that promote path independence also improve upwards generalization, while those that penalize it hurt this capability. Our findings suggest that path independent equilibrium models are a promising direction towards building general purpose learning systems whose test-time performance improves with larger compute.

References

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965. (Cited on 3, 6)

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. (Cited on 14)

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Neural Information Processing Systems*, 2019. (Cited on 2, 3, 5, 6)

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Neural Information Processing Systems*, 2020. (Cited on 3)

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Stabilizing equilibrium models by jacobian regularization. In *ICML*, 2021. (Cited on 5)

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. (Cited on 8)

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*, 2022. (Cited on 1, 4, 5, 9, 16, 17)

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. (Cited on 9)

Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 1965. (Cited on 3, 6, 17, 19)

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. 2019. URL https://openreview.net/pdf?id=HyzdRiR9Y7. (Cited on 8)

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SJg7KhVKPH. (Cited on 8)

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *ArXiv*, abs/1910.10073, 2020b. (Cited on 9)

Cristobal Eyzaguirre and Alvaro Soto. Differentiable adaptive computation time for visual reasoning. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 12817–12825, 2020. (Cited on 8)

Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048, 2017. (Cited on 8)

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Fixed point networks: Implicit depth models with jacobian-free backprop. *arXiv e-prints*, pages arXiv–2103, 2021. (Cited on 3)

Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is attention better than matrix decomposition? In *International Conference on Learning Representations (ICLR)*, 2021a. (Cited on 3)

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34, 2021b. (Cited on 3, 8)

Alex Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016. URL http://arxiv.org/abs/1603.08983. (Cited on 8)

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33:11984–11995, 2020. (Cited on 3)

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on 16, 17)

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021. (Cited on 1)

Zico Kolter, David Duvenaud, and Matthew Johnson. Tutorial: Deep implicit layers-neural odes, deep equilibirum models, and beyond, 2020. (Cited on 16, 19)

Alexandros Kouris, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas D Lane. Multi-exit semantic segmentation networks. *arXiv preprint arXiv:2106.03527*, 2021. (Cited on 9)

Stefanos Laskaridis, Alexandros Kouris, and Nicholas D Lane. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pages 1–6, 2021. (Cited on 9)

Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3193–3202, 2017. (Cited on 9)

Kaiqu Liang, Cem Anil, Yuhuai Wu, and Roger Grosse. Out-of-distribution generalization with deep equilibrium models. *Uncertainty and Robustness in Deep Learning, ICML 2021*. (Cited on 24)

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091. PMLR, 2018. (Cited on 2)

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989. (Cited on 5, 17, 18)

Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and X. Xiao. Eignn: Efficient infinite-depth graph neural networks. In *NeurIPS*, 2021a. (Cited on 3)

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*, 2020. (Cited on 9)

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. Faster depth-adaptive transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13424–13432, 2021b. (Cited on 8)

James L McClelland and David E Rumelhart. *Explorations in parallel distributed processing: A handbook of models, programs, and exercises*. MIT press, 1989. (Cited on 2)

John P Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *International Conference on Machine Learning*, pages 7721–7735. PMLR, 2021. (Cited on 7)

Mark Neumann, Pontus Stenetorp, and Sebastian Riedel. Learning to reason with adaptive computation. *arXiv preprint arXiv:1610.07647*, 2016. (Cited on 8)

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021. (Cited on 1, 9)

Max Revay, Ruigang Wang, and Ian R Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020. (Cited on 3)

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. URL http://arxiv.org/abs/1602.07868. (Cited on 6, 16, 17, 18, 19)

Jürgen Schmidhuber. Self-delimiting neural networks. *arXiv preprint arXiv:1210.0118*, 2012. (Cited on 8)

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. The right tool for the job: Matching model and instance complexities. *arXiv preprint arXiv:2004.07453*, 2020. (Cited on 9)

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Arpit Bansal, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. Datasets for studying generalization from easy to hard examples. *arXiv preprint arXiv:2108.06011*, 2021a. (Cited on 2, 4, 16, 17)

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34, 2021b. (Cited on 1, 2, 4, 9, 18)

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018. (Cited on 1)

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995. (Cited on 2)

Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991. (Cited on 1)

Luca Soldaini and Alessandro Moschitti. The cascade transformer: an application for efficient answer sentence selection. *CoRR*, abs/2005.02534, 2020. URL https://arxiv.org/abs/2005.02534. (Cited on 9)

Bharath K Sriperumbudur and Gert RG Lanckriet. On the convergence of the concave-convex procedure. In *Nips*, volume 9, pages 1759–1767. Citeseer, 2009. (Cited on 1)

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. (Cited on 9)

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. (Cited on 3, 8)

Qing-Guo Wang, Tong Heng Lee, and Chong Lin. Global stability of limit cycles. In *Relay Feedback*, pages 57–83. Springer, 2003. (Cited on 1)

Tiancai Wang, Xiangyu Zhang, and Jian Sun. Implicit feature pyramid network for object detection. *arXiv preprint arXiv:2012.13563*, 2020. (Cited on 3)

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. (Cited on 9)

Ezra Winston and J Zico Kolter. Monotone operator equilibrium networks. *arXiv preprint arXiv:2006.08591*, 2020. (Cited on 3)

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. (Cited on 17)

Qunliang Xing, Mai Xu, Tianyi Li, and Zhenyu Guan. Early exit or not: Resource-efficient blind quality enhancement for compressed images. In *European Conference on Computer Vision*, pages 275–292. Springer, 2020. (Cited on 9)

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33: 18330–18341, 2020. (Cited on 9)

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

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. $||\frac{\partial \mathrm{FIX}(\boldsymbol{x},\boldsymbol{z}_0)}{\partial \boldsymbol{z}_0}||_2$) 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 [7]) 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 FPA 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 the

[7] 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.

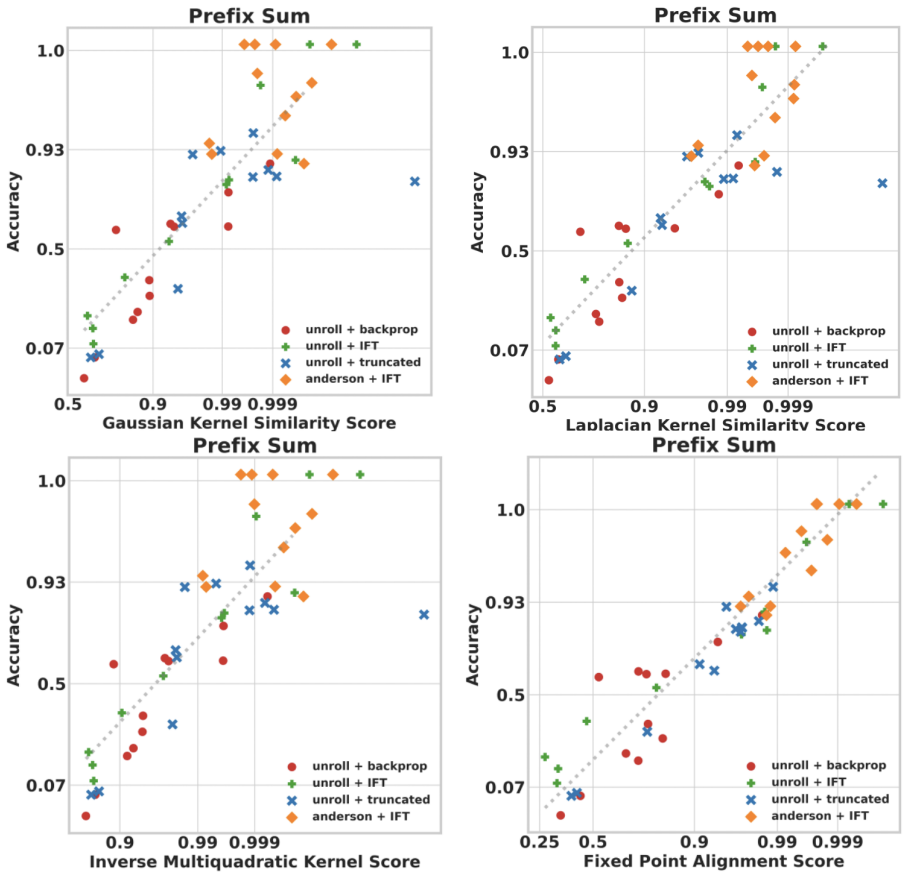


Figure 6: **Alternatives to Cosine Similarity in FPA Score:** Replacing cosine similarity (right bottom) with alternative kernels like Gaussian (left top), Laplacian (right top) and inverse multiquadratic kernsl yields qualitatively similar results. This demonstrates that the takeaways from our experiments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

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 theorical 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.

B Experimental Details

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

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.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 DEQ 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 FPA 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 FPA 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 DEQs needed a longer training time for their validation accuracy to improve. All DEQ 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 DEQs, we do rollouts for 32-iterations. For DEQ 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 DEQ. We found that weight normalization helps stabilize the training, especially DEQ 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 training DEQ 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 FPA 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 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 FPA scores and accuracy computed on 500 examples. We provide pseudocode in Algorithm Box 2

Algorithm 2 Stress testing FPA 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
  $\boldsymbol{z}_1 = \mathrm{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}, \boldsymbol{x})$
  ▷ Clone $\boldsymbol{z}_1$ and Enable gradients on $\boldsymbol{z}_1$
  for trials from 1 to N **do**
    $\boldsymbol{z}_2 = \mathrm{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}_1, \boldsymbol{x})$
    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 11th and 14th 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 12th and 18th 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 DEQ 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 FPA scores on 300 examples for the prefix-sum dataset validation splits of lengths 16, 32, 64, 128 and 256. We grouped each FPA 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 FPA 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.

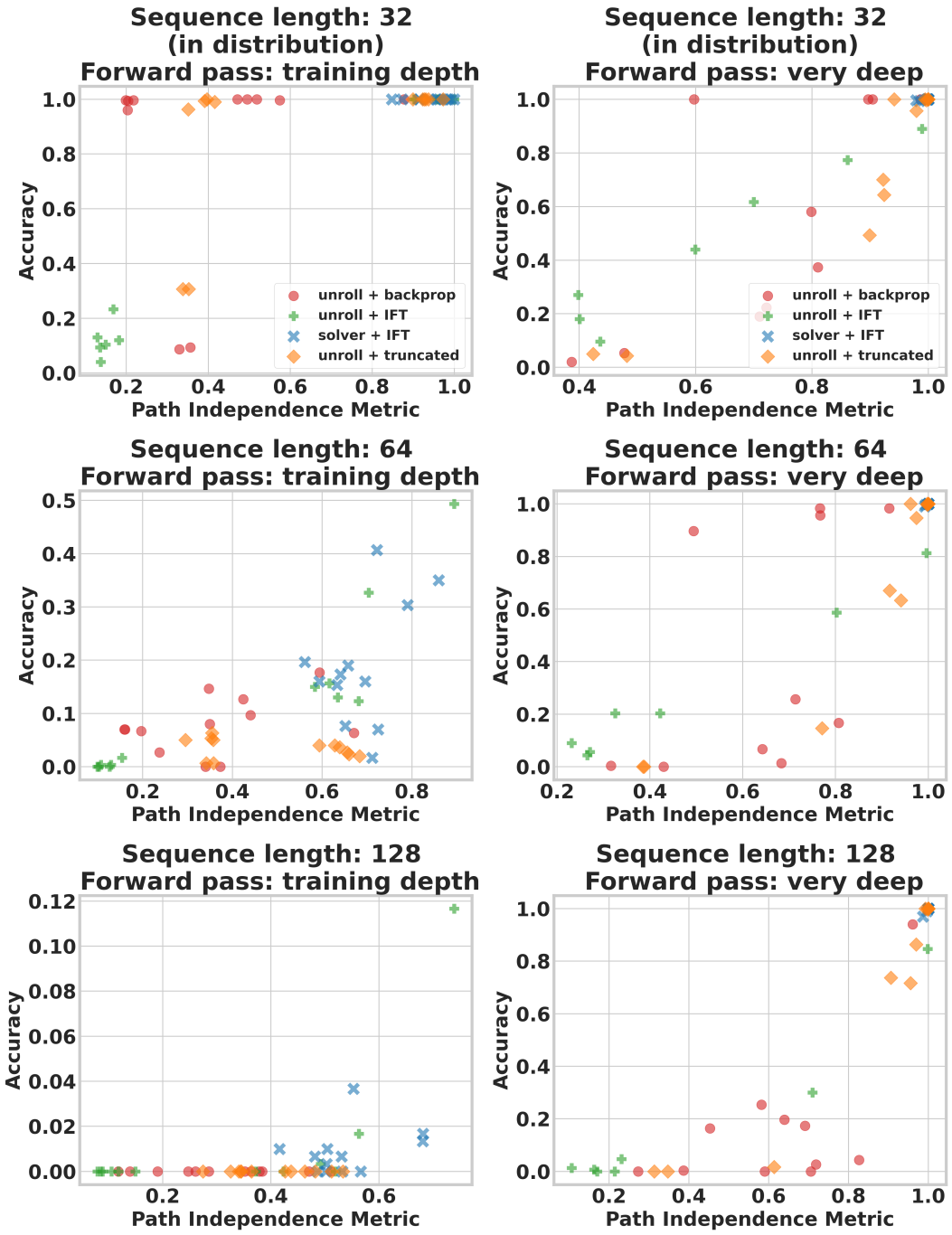


Figure 7: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in 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.

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 FPA 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 FPA 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. FPA 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 FPA 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.

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.

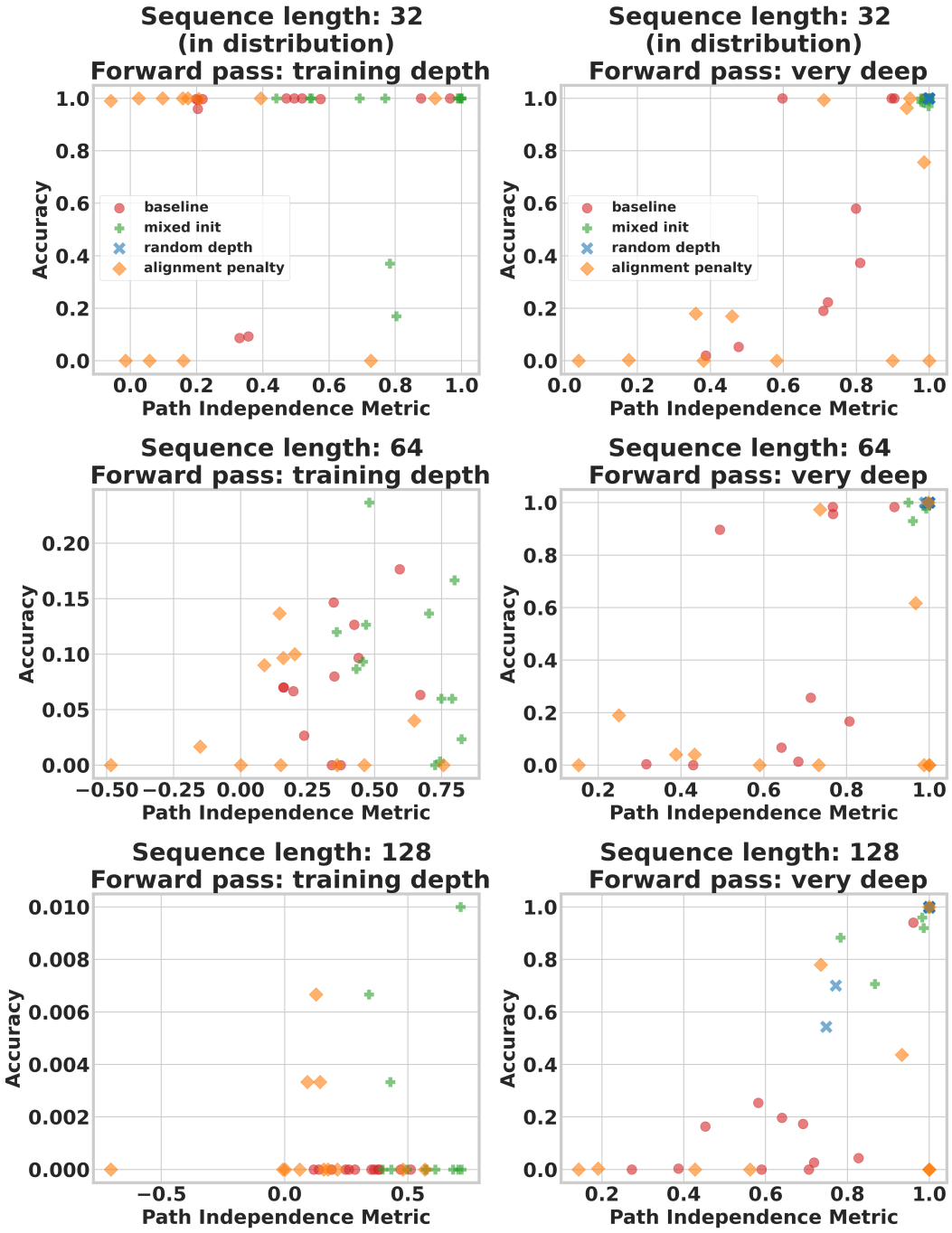


Figure 8: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in 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.

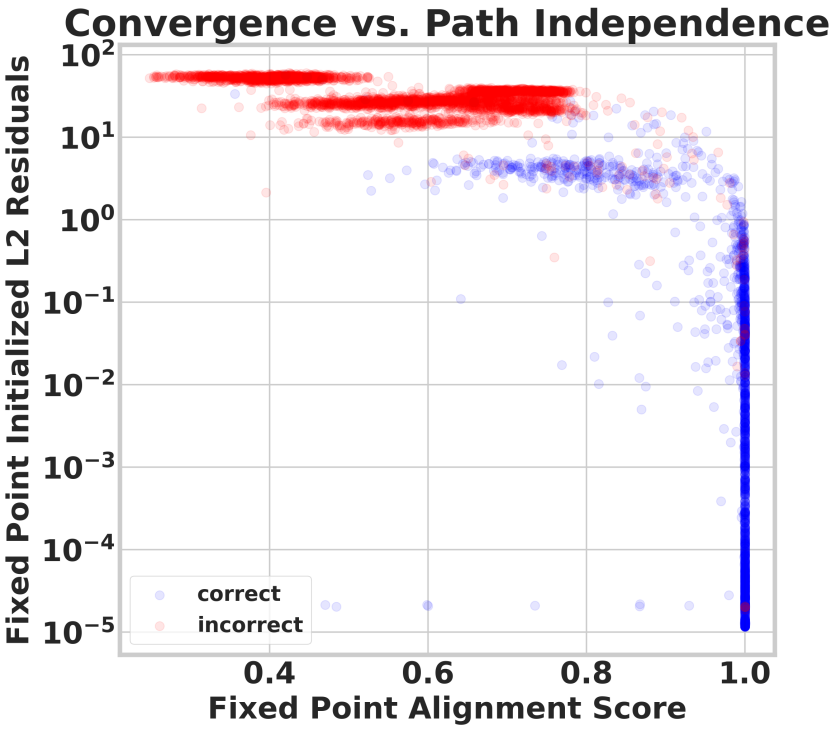


Figure 9: **Per-Instance Test-time Convergence vs. Path Independence**: The per-instance FPA score vs. convergence (measured in terms of the L2 distance between the last two root solver iterates). Each example is labelled based on whether the network's prediction on it was correct (blue) or not (red). 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 FPA values, the converse is not necessarily true: there are many samples on which convergence is poor, yet the FPA score is high.

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 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.

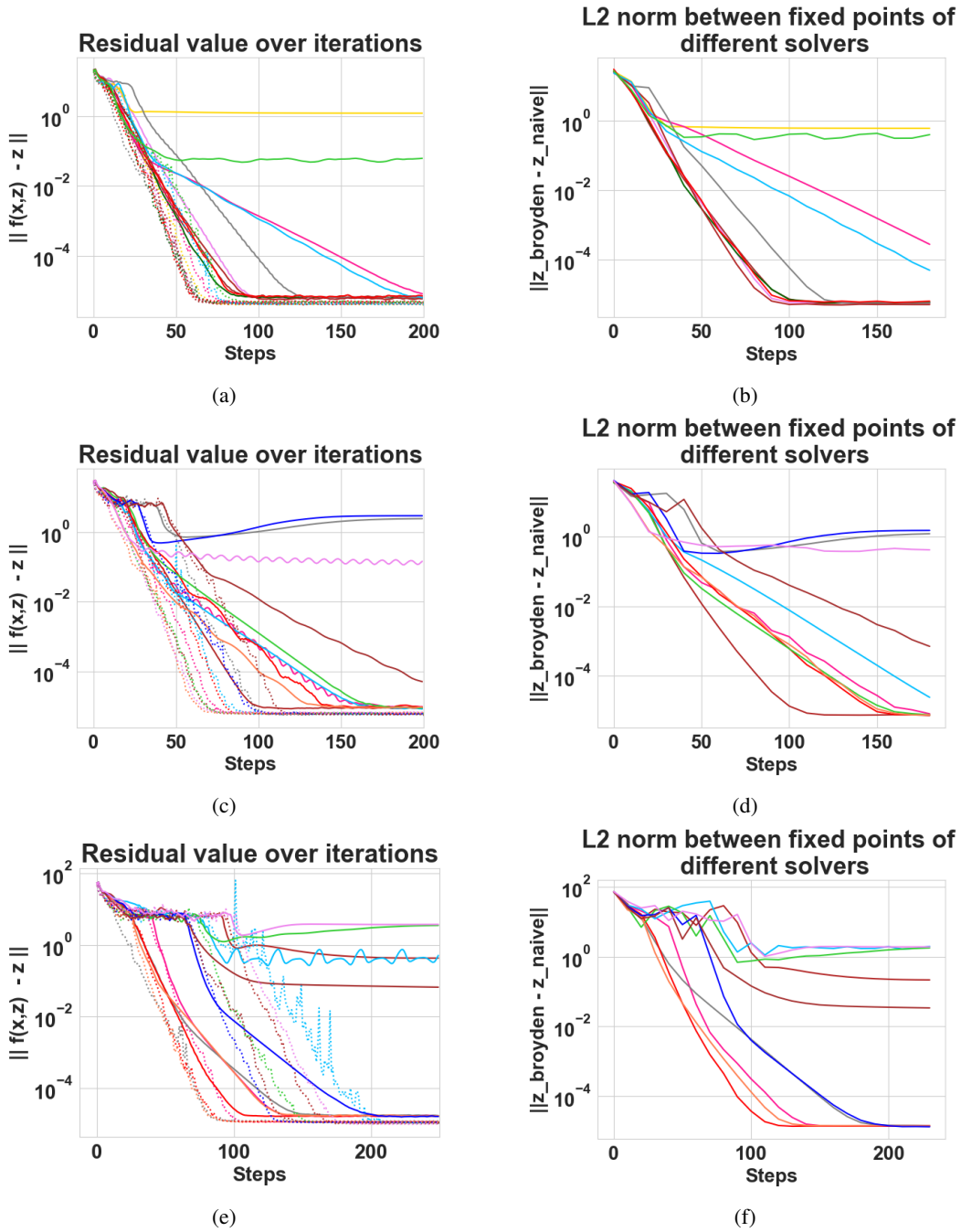


Figure 10: Different solvers display differing asymptotic behaviour but still achieve good upwards generalization (**Left column**) We display plots for the values of $||f(x,z) - z||_2$ over multiple solver steps for Broyden's method (dotted lines) and naive fixed point iterations (solid lines). (**Right 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

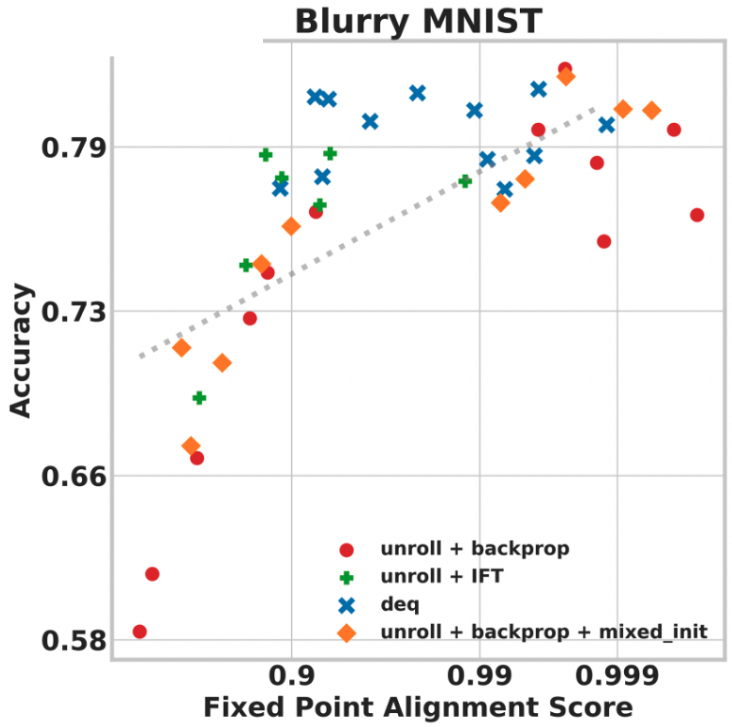


Figure 11: **BlurryMNIST results:** Path independence and generalization correlate on the BlurryMNIST dataset. This task involves training an image classifier on MNIST digits corrupted with small degrees of Gaussian blur, and testing the performance on significantly more corrupted ones.