FINDING ALGORITHMS IN NEURAL NETWORKS: CONVERGENCE BETWEEN DISTRIBUTED ALIGNMENT SEARCH AND SPEED OF TRANSFER LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

How can we identify the algorithm that a neural network learned on a task? One behavioral method, transfer learning analysis (TLA), is to read it off how quickly it learns related tasks. One mechanistic method, distributed alignment search (DAS), is to read it off from interventions on the network's internal structure. Whether these two methods converge is as of yet untested. We find that, on toy arithmetic and vision tasks, the algorithms identified by DAS and TLA mostly agree: the internal structure of neural networks predicts how quickly they will transfer to a related task (and conversely, speed of transfer predicts internal structure). Crucially, only DAS with non-linear localization agrees with TLA in our vision task. This means that TLA isn't merely a description of the network's behavior. DAS and TLA are two convergent sources of evidence for identifying the algorithm implemented by a neural network.

1 INTRODUCTION

Algorithms transform inputs into outputs through a series of operations on intermediate representations. Because they abstract away many of the details of their implementation, they offer simple perspectives on complex systems, and therefore might be useful for understanding neural networks. In this paper, we compare two approaches to attributing algorithms to neural networks: a behavioral method which looks at how quickly a network learns related tasks (Morrison et al., 2025), and a mechanistic method, distributed alignment search (DAS), which performs interventions on the network's internal structure (Geiger et al., 2024).

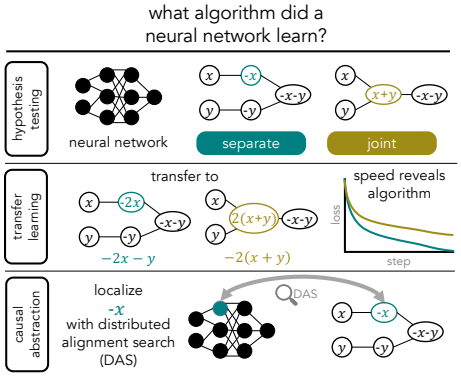

Figure 1: **Experimental setup.** A model can learn $-x-y$ via *separate* algorithm $(-x)+(-y)$ or via a *joint* algorithm $-(x+y)$ (top panel). We ask whether localizing the *separate* variable $-x$ with DAS (bottom panel) predicts how quickly it will transfer to $-2x-y$ where we edit the $-x$ variable (vs. $-2(x+y)$ where we edit the $x+y$ variable) (middle panel).

Behavioral approaches are standard in cognitive psychology (Marr, 1982). They use behavior to work inwards to an algorithm. Examples of behavior are performance, generalization, error patterns, reaction times, self-reports, and task transfer. In this paper, we focus on task transfer. In particular, we use transfer learning analysis (TLA), which identifies algorithms in neural networks. TLA connects how easy it is to transform one program into another and how quickly we can learn a second program from an implementation of the first (Morrison et al., 2025). This raises the question, do neural networks reuse learned structure to transfer to new data? If so, then the speed at which a network transfers to new tasks can help us infer its internal structure.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

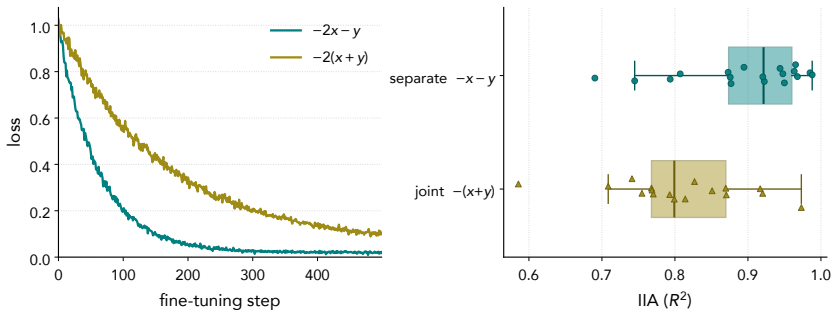

Figure 2: **Transfer learning analysis (TLA) and distributed alignment search (DAS) mostly agree.** Our results show that TLA, a behavioral method that identifies a network's algorithms by how quickly it learns related tasks, makes similar predictions to DAS, a mechanistic method that identifies a network's algorithms by intervening on its internal structure. Left: example of TLA. Faster transfer to $-2x-y$ means the network implements the *separate* algorithm. Right: distribution of IIA for the *separate* variable $-x$ for different networks, colored by whether TLA classifies them as separate or joint. Networks classified as separate have higher IIA ($t = 2.91$, $p = 0.006$ on paired t-test), meaning TLA and DAS agree.

The mechanistic approach is gaining traction in systems neuroscience and in machine learning (especially mechanistic interpretability) (Saphra & Wiegreffe, 2024). This approach looks for parts of the network that correspond to parts of an algorithm, and then uses that algorithm to explain its behavior. We focus on distributed alignment search (DAS), which localizes causal variables in a neural network through interventions on the its internal structure (Geiger et al., 2024). One goal of mechanistic interpretability is to predict how models behave in novel situations. Does the internal structure of the network predict how quickly it will transfer to related tasks?

To answer the above questions, we design controlled experiments that compare TLA and DAS in arithmetic and vision domains (illustrated in Figure 1). In particular, we train models on synthetic data generated by a program with known intermediate variables. We then compare the predictions made by TLA and DAS, which perform interventions at different stages. TLA intervenes on the data-generating program, replacing one or both intermediate variables, and measures how quickly the model learns the counterfactual task. DAS intervenes on the network internals, interchanging activations from different inputs, and measures whether parts of the network correspond to parts of the algorithm. We seek to understand whether localizing a variable in a neural network predicts how quickly the model transfers to a task where that variable was manipulated, and *vice versa*.

Our results, summarized in Figure 3, show that TLA and DAS mostly support each other. That is, TLA and DAS often predict the same algorithms for the same neural networks in our arithmetic and vision experiments. This is reassuring for identifying algorithmic implementation: the speed at which a neural network adapts to a new task predicts its internal structure. In turn, our results highlight the value of mechanistic analysis in evaluation (McCoy et al., 2019; Pavlick, 2023; Leshinskaya et al., 2025), since the internal structure of a neural network predicts its downstream behavior, in this case on how quickly it learns related tasks .

Crucially, in the vision task, DAS only agrees with TLA when parameterized with non-linear localization. This finding provides evidence against the linear representation hypothesis (LRH), which claims that neural networks represent intermediate computations in linear subspaces of their activations (Park et al., 2023). It also suggests that TLA can play an important role in guiding DAS by providing an additional source of evidence for identifying the algorithm implemented by a neural network.

2 METHODS

In this paper, we compare a behavioral method, TLA, and a mechanistic method, DAS, in identifying the algorithm implemented by a neural network. We ask whether, when presented with a set of hypothesized algorithms for a neural network, the two methods agree on the best interpretation.

Our experimental paradigm works as follows, illustrated in Figure 1. We generate a task through a data-generating process (DGP): inputs are mapped to an output through intermediate variables, and an algorithm specifies which intermediate variables are computed and how they are combined. The two methods we compare use the DGP differently. TLA manipulates it, editing an intermediate variable, and asks which manipulated task the network learns faster. DAS treats the DGP as a high-level causal model and asks whether there's an alignment between the network's activations and the DPG's intermediate variables. For simplicity of exposition, we'll use our arithmetic experiment as a running example in this section. However, both TLA and DAS are general methods, and we show they agree on a vision task as well (see Section 3.2).

2.1 Transfer Learning Analysis (TLA)

TLA is a behavioral method that identifies a neural network's algorithm by how fast it learns related tasks (Morrison et al., 2025). It connects learning speed and program transformation: how easy it is to transform one algorithm into another relates to how easy it is to learn a new algorithm given the current one.

	$-2x-y$	$-2(x+y)$
separate	fast	slow
joint	slow	fast

Table 1: Predicted transfer speed under the two hypotheses, with what each algorithm must change to reach the new task.

For instance, take a neural network trained to compute $-x-y$ from two numbers x and y. One possible algorithm it could implement is to negate each number separately, $-x$ and $-y$, and then add them together to form $(-x)+(-y)$; call this the *separate* hypothesis. Another algorithm it could implement is to first combine the two inputs by adding them $x+y$, and then negating this joint representation to compute $-(x+y)$; call this the *joint* hypothesis. Figure 1 illustrates the two hypothesized algorithms as causal models, with intermediate variables $-x$ and $-y$ in the *separate* algorithm and a joint intermediate variable $x+y$ in the *joint* algorithm.

How do we know which algorithm our neural network implements? Transfer learning analysis proposes that the complexity of transforming one program into another relates to the time it takes to learn a new algorithm from the current one. In our setting, this corresponds to the number of intermediate variables we need to change in order to implement the new algorithm, and how quickly a neural network converges on data generated by the new algorithm, respectively.

For example, consider transforming the intermediate variable in the *separate* algorithm by multiplying it by 2: $-x \rightarrow -2x$. This creates a new algorithm, $-2x-y$ (see Figure 1). Whereas we can transform the *separate* algorithm to form $-2x-y$ by changing one variable ($-x$), transforming the *joint* algorithm $-(x+y)$ into $-2x-y$ is a more complex operation. Transfer learning analysis proposes that the complexity of this transformation manifests in the speed of transfer learning: a neural network fine-tuned on $-2x-y$ will learn this task *faster* if it implements the *separate* algorithm, and *slower* if it implements the joint algorithm.

Conversely, we can transform the *joint* variable $x+y \rightarrow 2(x+y)$ to form a new algorithm, $-2(x+y)$. Whereas we need only manipulate one variable $x+y$ in the *joint* algorithm, transforming the *separate* algorithm into $-2(x+y)$ requires changing both $-x$ and $-y$. A network fine-tuned on $-2(x+y)$ will therefore learn the task *faster* if it implements the *joint* algorithm, and *slower* if it implements the *separate* algorithm.

TLA works by fine-tuning a network $\mathcal{N}$ on data generated by the transformed algorithms, such as $-2x-y$ and $-2(x+y)$, and measuring how quickly its loss converges on the transfer data. The faster a network transfers to the transformed algorithm, the more likely it implements the manipulated variable. We repeat this at six multipliers, giving six pairs of transformed algorithms, and let each pair vote for the algorithm whose manipulated variable it favors. Within a pair, if the network's loss on $-2x-y$ is lower than its loss on $-2(x+y)$ on more than 60% of the fine-tuning checkpoints at which the two are compared, that pair votes for the *separate* algorithm, and *vice versa* when the network transfers faster to $-2(x+y)$ than to $-2x-y$. If at least five of the six pairs vote the same way, TLA predicts that the network implements that algorithm. Otherwise there is insufficient evidence and TLA categorizes it as ambiguous.

Table 1 summarizes our controlled manipulations and predicted behavior for TLA on the arithmetic task $-x-y$. Following Morrison et al. (2025), we apply TLA to classify neural networks as *separate*, *joint*, or *ambiguous* through their speed of transfer to the corresponding tasks. We then ask whether these predictions agree with DAS, which identifies the algorithm implemented by a neural network through causal interventions to its internal structure.

2.2 Distributed Alignment Search (DAS)

DAS is a mechanistic method for identifying when a low-level causal model (in our case, a neural network) implements a high-level causal model (an algorithm) Geiger et al. (2024). In particular, DAS searches for evidence of *causal abstraction* between the low-level and high-level models (Geiger et al., 2021; 2025).

Causal abstraction An algorithm $\mathcal{A}$ is a causal abstraction of a neural network $\mathcal{N}$ when there's an alignment between the intermediate variables in the algorithm and the intermediate activations of the neural network. This alignment is formalized by $\omega : \mathbb{I}_{\mathcal{A}} \to \mathbb{I}_{\mathcal{N}}$, which maps interventions on the algorithm to interventions on the neural network[1]. That is, if we were to change the value of an intermediate variable X in algorithm $\mathcal{A}$ to x through an intervention $\mathcal{A}_{X \leftarrow x}$, then there should be a matching intervention we can apply to the neural network, $\mathcal{N}_{\omega(X \leftarrow x)}$.

Formally, algorithm $\mathcal{A}$ is a causal abstraction of neural network $\mathcal{N}$ if there exists an alignment ω such that for all interventions $X \leftarrow x$,

$$\mathcal{N}_{\omega(X \leftarrow x)} = \mathcal{A}_{X \leftarrow x} \tag{1}$$

Interchange intervention accuracy (IIA) In practice, it's infeasible to enumerate all possible interventions on a high-level algorithm. Hence, we empirically evaluate causal abstraction with *interchange intervention accuracy* (IIA), which uses input pairs b and s to construct interventions: swapping the activation values of a neural network processing b and s should correspond to swapping the value of an intermediate variable produced by the algorithm.

Formally, with respect to a dataset D of base and source inputs b and s, interchange intervention accuracy (IIA) is defined as

$$\text{IIA}(\mathcal{A}, \mathcal{N}; \omega, D) := \frac{1}{|D|} \sum_{b,s \in D} \mathbb{1}\left[\mathcal{N}_{\omega(X \leftarrow x_s)}(b) = \mathcal{A}_{X \leftarrow x_s}(b)\right] \tag{2}$$

The choice of counterfactual input b and s depends on the task, but in our setting we choose inputs that minimally vary in their value for the intermediate variable in the *separate* algorithm. We provide more details per experiment in Section 3.1 (arithmetic) and Section 3.2 (vision).

Distributed Alignment Search (DAS) Causal abstraction (and its empirical estimate IIA) compares a neural network $\mathcal{N}$ and a high-level algorithm $\mathcal{A}$ with respect to an alignment ω between their intermediate variables. How do we find ω? *Distributed alignment search* (*DAS*) is a method that parameterizes ω_θ with learnable parameters θ, and optimizes θ to maximize the IIA objective (Geiger et al., 2024).

Often, θ is a linear rotation R (Geiger et al., 2024). To perform an interchange intervention from a source input s to a base input b, we swap the first d dimensions of the rotated base and source pairs $R(b)$ and $R(s)$, and then apply the inverse of the rotation R^{-1} to compute a counterfactual activation. We follow the implementation of Wu et al. (2023), constructing a learned rotation matrix $R \in \mathbb{R}^{h \times d}$ and applying

$$\mathcal{N}_{\omega(X \leftarrow x_s)}(b) = h_b + R^{-1}(Rh_s - Rh_b) \tag{3}$$

where h_x is the activation of the neural network on input x at a chosen layer. We select the DAS hyperparameters d (intervention dimension, or how many linear subspaces we swap during an intervention) and l (layer where we perform the intervention) through a grid search.

[1]Usually a second parameter of causal abstraction, τ, maps from the output space of the algorithm to the output space of the neural network; for simplicity's sake, we assume they share the same output space, which is the case in our setting.

Non-linear interventions The linear transformation R in DAS is partially justified by the *linear representation hypothesis* (Alain & Bengio, 2016; Bolukbasi et al., 2021; Park et al., 2023), which states that neural networks represent high-level variables in linear subspaces. However, recent evidence suggests that in many cases, neural networks form variable representations best characterized by non-linear manifolds (Csordás et al., 2024; Engels et al., 2025; Bhalla et al., 2026). In light of this, we consider both linear and non-linear localization methods for DAS.

We follow Sutter et al. (2026) in constructing a non-linear DAS parameterization. We replace the DAS rotation R (see Equation 3) with RevNet, a small invertible neural network (Gomez et al., 2017). To select the output dimension of the RevNet network, d, and the layer at which it gets applied, l, we perform the same grid search as for linear DAS. Details on the RevNet architecture and hidden dimensions are provided in Appendix A.

2.3 Method Comparison

For a given neural network $\mathcal{N}$ and algorithm $\mathcal{A}$, the TLA and DAS estimate the extent to which $\mathcal{N}$ implements $\mathcal{A}$. In this paper, we compare how often these methods agree.

To do so, we take a set of neural networks that were trained on the same task but show different transfer learning behavior, and ask whether the mechanistic predictions made by causal abstraction match the behavior observed after fine-tuning on related tasks. For the sake of comparison, we need not assume a ground truth; that is, we don't train neural networks to implement any particular algorithm. DAS and TLA are methods for estimating the true underlying algorithm, and in this paper we compare their predictions.

We follow the experimental design of Morrison et al. (2025), which classifies neural networks by how quickly they transfer to related tasks (see Figure 1). The classification procedure is specific to each task and is given in Section 3.1 and Section 3.2. We ask whether the interchange intervention accuracy for the *separate* variable is a good predictor of whether the neural network is classified as *separate* by transfer learning analysis, and *vice versa*. Our hypothesis is that the higher the interchange intervention accuracy is for the separate variable, the more likely the neural network is to be classified as separate.

3 Experiments

We study two tasks, an arithmetic task and an image task, using networks trained by Morrison et al. (2025). Each task can be solved by at least two algorithms. While the algorithms generate the same outputs, they differ in their intermediate variables. We want to know how often the behavioral method and the mechanistic method attribute the same algorithm to the same network.

3.1 Experiment I: Arithmetic

In the arithmetic task (Figure 1), we train neural networks to take two numbers x and y, each drawn from $[-50, 50]$, and output $-x-y$ (we train on 1,000 pairs in total). A network can reach this output in two ways: it can first negate x and y separately $-x, -y$, and then compute $-x - y$ (*separate*), or it can first add the two numbers $x + y$, and then compute $-(x + y)$ (*joint*). The two algorithms generate identical outputs. To attribute one algorithm rather than the other, we must consider either its transfer behavior or its internal computation.

Models The arithmetic networks are the 100 feedforward ReLU networks of Morrison et al. (2025), with layer sizes $2 \to 8 \to 4 \to 8 \to 1$. They are trained by stochastic gradient descent on the mapping $(x, y) \to -x - y$, and any network with a loss above 0.05 is reinitialized and retrained, so all 100 solve the original task.

Identifying Algorithms with TLA We construct two transfer tasks: one where we transform $-x$ into $-2x$ in the *separate* algorithm, getting $-2x - y$, and one where we apply transform $x + y$ into $2(x + y)$ in the *joint* algorithm, getting $-2(x + y)$. We expect neural networks that represent x and y *separately* to transfer faster to the $-2x - y$ task; meanwhile, we expect networks that represent x and y *jointly* as $x + y$ to transfer faster to $-2(x + y)$.

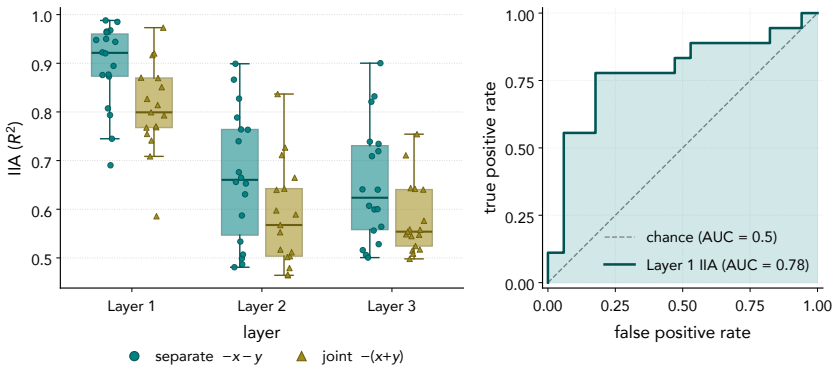

Figure 3: **Results for the arithmetic task.** Left: interchange intervention accuracy (IIA) for the variable $-x$ at each hidden layer, for models labeled $-x-y$ (*separate*, $n = 18$) and $-(x+y)$ (joint, $n = 17$) by transfer learning. Right: ROC curve for predicting the transfer label from layer 1 IIA, with $-x - y$ as the positive class (AUC 0.78).

Labels come from the transfer procedure of Morrison et al. (2025). Each network is fine-tuned on six pairs of new input-output mappings, and we record which member of each pair it learns faster. Each fine-tuning run uses 1,600 training and 400 validation points drawn from the same domain as training ($x, y \sim U(-50, 50)$), relabeled by the transformed algorithm, and runs for 75 epochs at batch size 5 (24,000 batches), with validation loss recorded every 20 batches. Within a pair we compare the two validation losses checkpoint by checkpoint: if the loss on the separate transform is lower on more than 60% of the compared checkpoints, the pair votes *separate*; if the loss on the joint transform is lower on more than 60%, it votes *joint*; between 40% and 60% the pair casts no vote. A network is labeled only when at least five of the six pairs agree, and is otherwise ambiguous. Of the 100 networks, 18 are labeled $-x - y$, 17 are labeled $-(x + y)$, and 65 are unlabeled and excluded from our analyses.

Identifying Algorithms with DAS Whereas TLA a model's algorithm through its behavior, DAS characterizes a model's algorithm through its internal structure. In the arithmetic task, this comes down to where the neural network implements the intermediate variables in the *separate* and *joint* tasks, as in Figure 1.

To localize the *separate* variable $-x$, we perform interchange interventions between source and base pairs (s, b) to form the counterfactual $-x_s - y_b$. Base and source inputs are drawn independently and uniformly from $[-50, 50]$ in each coordinate. We fit the alignment on 100,000 pairs at the output of each hidden layer and measure the error against the counterfactual on 1,000 held-out pairs. We expect that cleaner localization of $-x$ (higher IIA) predicts faster transfer to the *separate* $-2x - y$ task.

3.2 Experiment II: Vision

In the vision task (Figure 4), we train networks to classify synthetically generated images by a combination of their shape category, background color, shape color, and shape area.

Each image shows one of four shapes, an ellipse, a circle, a square, or a capsule, and carries three continuous latent variables: object color, background color, and area. The four shapes fall into two categories, with the ellipse and circle in category $+1$ and the square and capsule in category -1. The label is 1 when

$$\text{shape category} \times (\text{object color} + \text{background color} + \text{area}) \geq 0$$

and 0 otherwise. Because the shape enters the label only through its category, the two shapes within a category are interchangeable. A network can therefore treat each category as a unit, which we call a *joint* representation, or represent each shape on its own, which we call a *separate* representation (see Figure 4).

Models We analyze the CNN models trained by Morrison et al. (2025). Each network takes in an image of shape 64×64 and passes it through three convolutional layers with 16, 32, and 64 filters, each 3×3, with 2×2 max pooling after the first two, then a 128-node dense layer with ReLU and a sigmoid output.

Identifying Algorithms with TLA As illustrated in Figure 4, we create two transfer tasks, one which flips the *shape category* while keeping the latent variables the same and one which flips the *latent variables* while keeping the shape category the same. We expect that for models which implement *separate* detectors for each shape categories, flipping the shape category will be relatively easy to learn, since it amounts to flipping the shape category value for two shape detectors. On the other hand, for models which implement *joint* detectors for round versus square shapes, this transfer task is designed to be difficult: the joint algorithm must implement new variables to represent the transfer $+1$ and -1 categories.

Flipping the latent variables serves as a control. Since these variables are independent of the shape category, we expect both the *joint* and the *separate* networks to transfer equally quickly to this manipulation. We label networks as *separate* if they transfer faster to the *flipped shape category* task than to the *flipped latent variable* tasks, we label them as *joint* if it's the other way around, and we label them as *ambiguous* if the transfer results are mixed. Concretely, at each of the 250 checkpoints we compare the loss on the *flipped shape category* task against the three *flipped latent variable* tasks, scoring 1 if it exceeds every one of them by at least one standard deviation of their spread, 0 if it falls below their mean plus one standard deviation, and 0.5 otherwise. We average this score over the checkpoints up to the point where all four tasks have converged. A network is labeled *joint* if its average exceeds 0.6 and *separate* if it falls below 0.4. The intermediate band is *ambiguous*: these are networks whose shape-category transfer is neither reliably slower nor reliably faster than the latent-variable controls.

In total, from 362 neural networks, we classify 166 as joint, 36 as separate, and 160 as ambiguous. Because the separate group provides at most 28 usable networks, we subsample all three types to 28 each, giving a balanced set of 84 networks for analysis.

Identifying Algorithms with DAS We implement DAS to localize the shape category variable. In particular, we randomly generate pairs of shapes with different shape categories (-1 and $+1$) and the same latent values for the object color, background color, and area variables. An interchange intervention between these two inputs localizes the variable representing the shape category. We hypothesize that for networks to transfer quickly to the task which manipulates shape category, they must represent the shape category in a low-dimensional subspace of their activation space.

We consider two variants of DAS in the vision task, a *linear* DAS that learns a rotation matrix $R \in \mathbb{R}^{h \times d}$ to localize the shape category, and a *non-linear* DAS that implements a RevNet R following Sutter et al. (2026). We provide additional details, including hyperparameter selection and training details, in Appendix A.

4 Results

4.1 Results for Experiment I: Arithmetic

Figure 3 (left) shows IIA for the variable $-x$ at each hidden layer, for the 18 models labeled *separate* and the 17 labeled *joint* by TLA. The two groups differ in the predicted direction at every layer: models labeled *separate* localize $-x$ at higher IIA than models labeled *joint* (amir: REPORT T-TEST!). This suggests that models reuse their internal structure to transfer to new tasks: models with higher IIA for the separate variable $-x$ are significantly more likely to transfer quicker to the *separate* task $-2x - y$ than to the *joint* task $-2(x + y)$.

The same comparison can be read at the level of individual models. We take each model's layer 1 IIA as a score and its transfer label as the target, with $-x-y$ as the positive class. The score predicts the label above chance, with an AUC of 0.78 (Figure 3, right). Therefore, TLA and DAS support each other, attributing the same algorithm (between *separate* and *joint*) to the same neural networks.

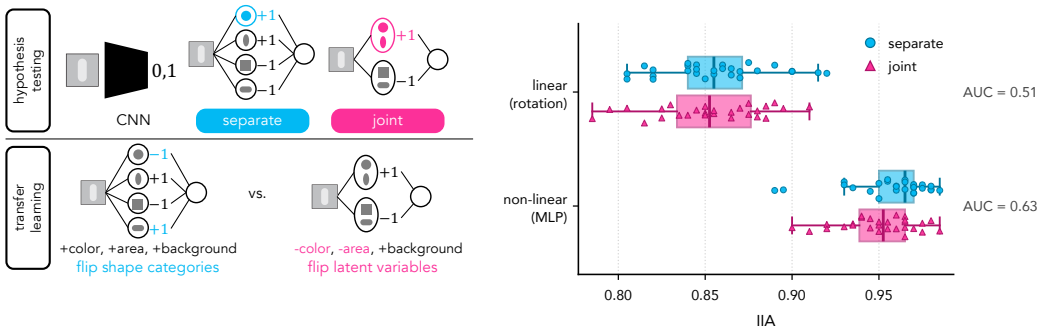

Figure 4: **Results for the vision task**. Left: experimental setup. We take neural networks trained to classify images by a combination of their shape category, color, area, and background. We consider speed of transfer to two tasks, one where we edit the shape category, and one where we edit the latent variables; we expect networks that implement *separate* shape detectors to transfer more quickly to the former. Right: IIA for localizing the *shape category* variable, using DAS with different localizations (linear vs. non-linear). Non-linear DAS localization improves IIA, and also agrees with TLA (Welch's t-test, $t = 2.40$, $p = 0.02$).

4.2 Results for Experiment II: Vision

We report the results for the vision task in Figure 4. First, we find that for the same layer (`relu3`) and interchange intervention dimension ($d = 1$), the non-linear localization method achieves higher IIA than linear localization. This suggests that, for the CNNs we analyze, the shape category is implemented in a non-linear subspace. Moreover, importantly, only non-linear localization predicts downstream task transfer: whereas the *separate* and *joint* models have the same IIA under linear DAS (IIA$= 0.85$, $t = 0.37$, $p = 0.72$), non-linear DAS achieves statistically significantly higher IIA for *separate* than *joint* models when localizing the *shape category* (IIA$= 0.96$ vs. 0.95, $t = 2.40$, $p = 0.02$)[2] That is, TLA and DAS only agree when the localization method used by DAS is effective. This means that TLA can provide a useful behavioral signal in guiding DAS.

We additionally compare DAS to probing, a correlational method that trains classifiers to decode the shape category of the input from the neural network's activation over the input. We train two probing variants, one over the entire activation, and one projected through the trained DAS subspace. Interestingly, we find that projecting through the DAS subspace only helps non-linear probes (accuracy dips from 0.86 to 0.71 for linear probes, but increases from 0.88 to 0.95 for non-linear probes). Nevertheless, we find that although probes can successfully decode the shape category from a network's activations (categorization accuracy of 0.95), their predictions don't align with the *separate* vs. *joint* categories from TLA. This result confirms earlier findings by Morrison et al. (2025), who find that correlational methods for identifying algorithms didn't agree with TLA predictions. See Appendix B for other mechanistic methods we considered; none agreed with TLA.

5 Discussion

In this paper, we show that TLA and DAS largely agree on toy arithmetic and vision tasks: more often than not, they attribute the same algorithm to the same neural network. These results bridge two notions of algorithmic implementation, one grounded in learning and representations and the other grounded in causal abstraction. TLA provides evidence on the basis that how easy it is to transform one algorithm into another determines how fast one can learn a second algorithm after knowing the first. DAS provides evidence that the intermediate computation performed by the neural network maps to an intermediate variable in the algorithm, and that this alignment holds upon interventions on the algorithmic variable.

[2]We use a Welch's t-test after removing outliers using standard interquartile range outlier detection.

One question that DAS raises is, how much IIA is enough? That is, when we can't achieve a perfect causal abstraction, how do we decide which algorithm a model implements? TLA can help guide DAS by providing an additional source of evidence. For example, in the vision experiment, we originally looked for the distinctions found by transfer learning in linear subspaces of the network, and couldn't find them. But because transfer learning gave us independent reason to think the distinction was real, we then searched non-linear subspaces of varying dimensions using a range of techniques and strategies until we found a partial alignment. Without the transfer learning results, we might not have looked beyond linear subspace and we might not have tried so many strategies. Conversely, it can tell us when to stop. In the algebraic task, moving to non-linear raises accuracy but dissolves the distinction found by transfer learning, indicating that linear subspaces were the right stopping point.

Ultimately, even in toy settings, DAS and TLA don't always agree. On the one hand, DAS finds high IIA for variables from the separate algorithm in networks that transfer more quickly to the joint transfer task. On the other hand, TLA depends on the decision of how to determine transfer speed, and doesn't always conclusively identify the algorithm implemented by a neural network (e.g., in the vision experiment, 160 of 362 models were labeled as ambiguous). It's possible, if not likely, that neural networks simultaneously implement multiple algorithms for the same task (Zhong et al., 2023). Looking for parallel implementation of multiple algorithms is an interesting area for future research in extending DAS and TLA.

6 Related Work

Causal Abstraction. Mechanistic interpretability explains a network's behavior by identifying the internal components that produce it (Olah et al., 2020; Elhage et al., 2021; Saphra & Wiegreffe, 2024; Sharkey et al., 2025). Recent work in mechanistic interpretability focuses on causal methods for identifying components that explain a behavior (Vig et al., 2020; Meng et al., 2022; Wang et al., 2022; Mueller et al., 2026), with causal abstraction providing a formal framework for when a network implements an algorithm (Geiger et al., 2021; 2025). DAS is a method for finding a causal abstraction between a hypothesized algorithm (high-level causal model) and a neural network (Geiger et al., 2024; Wu et al., 2023). Although it typically searches linear subspaces, following the linear representation hypothesis (Alain & Bengio, 2016; Park et al., 2023), some variables are represented on non-linear manifolds (Csordás et al., 2024; Engels et al., 2025; Bhalla et al., 2026; Wurgaft et al., 2026).

Transfer learning analysis. Behavioral methods infer an algorithm from what a system does rather than from its parts (Marr, 1982). This approach is standard in cognitive science (Binz & Schulz, 2023; Leshinskaya et al., 2025) and in analyses of how neural networks err and generalize (Lake & Baroni, 2018; McCoy et al., 2019). Transfer is a classic signature of shared structure (Singley & Anderson, 1989), and theory for deep linear networks ties learning speed and transfer to internal structure (Saxe et al., 2013; Lampinen & Ganguli, 2018; Hu et al., 2025). TLA uses transfer speed to decide between algorithms that produce the same outputs (Morrison et al., 2025), which suggests that neural networks reuse internal structure to learn new tasks (Jain et al., 2024; **?**). Like TLA, Rozner & Shain (2026) read representations off learning, measuring how fine-tuning on one example spreads to others. Most closely related, Huang et al. (2025) show that causal variables found with interventions predict out-of-distribution behavior. We extend this bridge from predicting outputs to predicting learning speed, and in turn use TLA to guide DAS.

References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Usha Bhalla, Thomas Fel, Can Rager, Sheridan Feucht, Tal Haklay, Daniel Wurgaft, Siddharth Boppana, Matthew Kowal, Vasudev Shyam, Jack Merullo, et al. Do sparse autoencoders capture concept manifolds? *arXiv preprint arXiv:2604.28119*, 2026.

Marcel Binz and Eric Schulz. Using cognitive psychology to understand gpt-3. *Proceedings of the National Academy of Sciences*, 120(6):e2218523120, 2023.

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for bert. *arXiv preprint arXiv:2104.07143*, 2021.

Róbert Csordás, Christopher Potts, Christopher D. Manning, and Atticus Geiger. Recurrent neural networks learn to store and generate sequences using non-linear representations. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 248–262, 2024.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021.

Joshua Engels, Eric J. Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not all language model features are one-dimensionally linear. In *International Conference on Learning Representations*, 2025.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. *Advances in neural information processing systems*, 34:9574–9586, 2021.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In *Causal Learning and Reasoning*, pp. 160–187. PMLR, 2024.

Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, et al. Causal abstraction: A theoretical foundation for mechanistic interpretability. *Journal of Machine Learning Research*, 26(83):1–64, 2025.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Michael Y Hu, Jackson Petty, Chuan Shi, William Merrill, and Tal Linzen. Between circuits and chomsky: Pre-pretraining on formal languages imparts linguistic biases. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9691–9709, 2025.

Jing Huang, Junyi Tao, Thomas Icard, Diyi Yang, and Christopher Potts. Internal causal mechanisms robustly predict language model out-of-distribution behaviors. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 25791–25812. PMLR, 13–19 Jul 2025. URL `https://proceedings.mlr.press/v267/huang25af.html`.

Samyak Jain, Robert Kirk, Ekdeep Singh Lubana, Robert Dick, Hidenori Tanaka, Tim Rocktäschel, Edward Grefenstette, and David Krueger. Mechanistically analyzing the effects of fine-tuning on procedurally defined tasks. In *International Conference on Learning Representations*, volume 2024, pp. 9944–10015, 2024.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pp. 2873–2882. PMLR, 2018.

Andrew K Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv preprint arXiv:1809.10374*, 2018.

Anna Leshinskaya, Taylor Webb, Ellie Pavlick, Jiahai Feng, Gustaw Opielka, Claire Stevenson, and Idan A Blank. Cognitively inspired interpretability in large neural networks. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 47, 2025.

David Marr. *Vision*. W. H. Freeman, 1982.

R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 3428–3448, 2019.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

John Morrison, Nikolaus Kriegeskorte, and Benjamin Peters. Using transfer learning to identify a neural system's algorithm. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 47, pp. 3802–3809, 2025.

Aaron Mueller, Jannik Brinkmann, Millicent Li, Samuel Marks, Koyena Pal, Nikhil Prakash, Can Rager, Aruna Sankaranarayanan, Arnab Sen Sharma, Jiuding Sun, et al. The quest for the right mediator: Surveying mechanistic interpretability for nlp through the lens of causal mediation analysis. *Computational Linguistics*, 52(1):331–378, 2026.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023.

Ellie Pavlick. Symbols and grounding in large language models. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 381(2251):20220041, 2023.

Joshua Rozner and Cory Shain. Perturbation: A simple and efficient adversarial tracer for representation learning in language models. *arXiv preprint arXiv:2603.23821*, 2026.

Naomi Saphra and Sarah Wiegreffe. Mechanistic? In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 480–498, 2024.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeff Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Bloom, et al. Open problems in mechanistic interpretability. *arXiv preprint arXiv:2501.16496*, 2025.

Mark K Singley and John Robert Anderson. *The transfer of cognitive skill*. Number 9. Harvard University Press, 1989.

Denis Sutter, Julian Minder, Thomas Hofmann, and Tiago Pimentel. The non-linear representation dilemma: Is causal abstraction enough for mechanistic interpretability? *Advances in Neural Information Processing Systems*, 38:149455–149503, 2026.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401, 2020.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah Goodman. Interpretability at scale: Identifying causal mechanisms in alpaca. *Advances in neural information processing systems*, 36:78205–78226, 2023.

Daniel Wurgaft, Can Rager, Matthew Kowal, Vasudev Shyam, Sheridan Feucht, Usha Bhalla, Tal Haklay, Eric Bigelow, Raphael Sarfati, Thomas McGrath, et al. Manifold steering reveals the shared geometry of neural network representation and behavior. *arXiv preprint arXiv:2605.05115*, 2026.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in neural information processing systems*, 36:27223–27250, 2023.

AI USE STATEMENT

In this work, we used generative AI tools for developing experiment code, drafting sections of the appendix, and searching for related literature. We have not used generative AI tools for coming up with research questions or experiment ideas, nor for writing the main body of the text. We have reviewed all AI-assisted work. We read over and tested the code it generated, and closely reviewed the text it drafted. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will include a code repository in the final version of the paper, including code for re-generating the data and re-training models with fixed random seeds. The arithmetic experiments can be recomputed on a CPU. The vision tasks require a GPU, and are estimated to take one to three days to reproduce.

A DISTRIBUTED ALIGNMENT SEARCH IMPLEMENTATION

This appendix collects the analyses behind Section 4 and the alternatives we tried, so that the null for linear methods and the non-linear signal can be checked against everything that was run. Table 2 lists each vision analysis with the models it was computed on, how well it localized shape, and whether it separated joint from separate models. The analyses were run at different stages of the project on different subsets of the trained population, and the table states the population for each. The subsections below describe each analysis.

A.1 TRAINING AND EVALUATION

In the vision task, DAS asks whether the shape category lives in a small subspace of a chosen layer. We freeze the network and learn a rotation of that layer that picks out a d-dimensional subspace (Equation 3). Given a base image and a source image, we run both through the network, replace the base's coordinates inside the subspace with the source's, and finish the forward pass. If the subspace carries the shape category, the swap should move the output to the value the network would give if the base image had the source's category while keeping its own color, background, and area. That target is the counterfactual: the source's shape category times the sum of the base image's object color, background color, and area. We read the intervention in two ways: interchange intervention accuracy (Equation 2), the fraction of image pairs whose swapped output matches the counterfactual label, and directional accuracy, the fraction whose output logit moves in the correct direction.

DAS runs in two phases. We first train the rotation on one cross-category shape pair and record its cross-train accuracy, then freeze the rotation and apply it unchanged to a different, held-out pair. Transfer accuracy tests whether the subspace holds a general shape-category direction or one specific to the training pair: a general direction carries over to a held-out pair, a pair-specific one does not. In the first experiment we train on ellipse and capsule and transfer to circle and capsule. In the second, we train on circle and square and transfer to circle and capsule, ellipse and square, and ellipse and capsule.

We train with a mean squared error loss on the network's logit calibrated to the counterfactual, with a binary cross-entropy loss on the counterfactual label as an alternative, a learning rate of 10^{-3}, a batch size of 64, and at most 5000 steps with early stopping, which in practice stops well before the cap. We intervene at relu2, conv3, relu3, and relu4, whose flattened activations have 32,768, 16,384, 16,384, and 128 dimensions, and sweep the subspace dimension d at each layer (Appendix A.4). The task, the stimuli, and the four shapes are described in Section 3.2 and illustrated in Figure 4.

A.2 SOURCE-IMAGE SAMPLING

Which source image should we pair with each base image? Section 3.2 describes the three schemes we tried. Minimal-pair sampling brings the cross-train step to ceiling: every model passes the cross-train threshold in both intervention directions (67 of 67 at conv3, dimension 16), against 0.985 and 0.910 by direction under random sampling. We therefore use minimal pairs for the transfer figures,

Table 2: Analyses behind the vision results. n is joint/separate/intermediate, and a dash means intermediate models were excluded. A slash separates joint from separate; where a row covers several settings, values follow the order of the settings. AUCs take separate as the positive class, so above 0.5 means separate models scored higher.

Analysis	Setting	n (j/s/i)	Localization	Joint vs separate
Linear DAS				
Random pairs	conv3, d=16	22/22/23	pass rate 0.985, 0.910 by direction	joint − separate +0.011
Minimal pairs	conv3, d=16	22/22/23	67 of 67 pass	joint − separate +0.003
Per-pair transfer	random pairs, held-out pairs	28/22/28	transfer 0.70 to 0.72 (circle to capsule), 0.39 (ellipse to capsule)	overlap, every pair
Generated pairs	conv3, pool2, sweep over d	28/22/28	cross-train 0.932, 0.908; transfer 0.336 to 0.420, 0.238 to 0.352	overlap, every d
All four latents	relu2, d=4, 16	28/22/28	shape IIA 0.319, 0.552	AUC 0.46, 0.50
All four latents	relu3, d=4, 16, 64	28/22/28	shape IIA 0.792, 0.878, 0.882	AUC 0.46, 0.49, 0.42
All four latents	relu4, d=4, 16	28/22/28	shape IIA 0.850, 0.851	AUC 0.53, 0.53
Dimension sweep	conv3, d=1 to 64	10/8/8	transfer 0.53 to 0.58 at d=1, 0.66 at d=4	converge from d=4
Dimension sweep	relu3, d=256	137/30/133	cross-train 0.531	0.529 / 0.546
Logit-difference readout	circle and square	28/22/–	same result as the accuracy analysis	no difference
Non-linear DAS (RevNet)				
Sweep	relu2 to relu4, d=1 to 64, best B, w per cell	28/22/28	IIA 0.81 to 0.89 (relu2), 0.80 to 0.87 (relu3), 0.98 (relu4)	joint leads in 23 of 48 runs, separate in 25
Recovery at relu4	d=1, B=1, w=32	28/22/28	IIA 0.942	AUC 0.63
Low dimension	relu3, d=1, 2, B=1, w=32	28/22/28	IIA 0.824, 0.870	AUC 0.33, 0.37
Other mechanistic methods				
Steering (difference of means)	relu1 to relu4	137/30/100	$\|\Delta\text{logit}\|$ 3.1 to 20.2; direction 0.53 to 0.75	overlap except relu1
Single feature map patch	conv1 to conv3	135/30/133	transfer ratio 0.97 to 0.66 with depth	AUC 0.44 to 0.50
Probing (linear, MLP)	16-dim DAS subspace	28/22/–	shape accuracy, linear 0.749 / 0.750, MLP 0.927 / 0.922	88 of 90 cells within 2 SE
PCA, raw activations	relu4, controlled set	5/5/5	ellipse to circle separation 0.635 / 0.674	AUC 0.72

and random sampling for the pair-difficulty analysis, where the per-pair differences are still visible. The joint-minus-separate transfer gap is $+0.003$ under minimal pairs and $+0.011$ under random sampling (Table 2).

Transfer difficulty depends on the shape pair, not on the model type. Under a rotation trained on circle and square and evaluated on held-out pairs with random sampling, circle to capsule transfers at about 0.70 to 0.72, while ellipse to capsule reaches only about 0.39, and joint and separate overlap on every pair (Table 2). Directional accuracy tells the same story in both intervention directions.

Generated minimal pairs are the strictest control: the source image is rendered with the same color, background, and area as the base, so the pair differs in shape alone. Joint and separate still overlap at every dimension, at conv3 and at pool2 (Table 2). At conv3 cross-train accuracy reaches 0.932 at dimension 16 and transfer accuracy rises from 0.336 to 0.420 across dimensions; at pool2, cross-train reaches 0.908 and transfer rises from 0.238 to 0.352.

A.3 Layer Selection

We chose the intervention layer from the run in which a single rotation localizes all four latents at once. Shape IIA is below or near chance at relu2 (0.319 at dimension 4, 0.552 at dimension 16) and reaches 0.79 to 0.88 from relu3 onward (Table 2).

Joint, separate, and intermediate models overlap at every layer and subspace dimension. The largest mean gap between any two types is 0.015, and between joint and separate it is 0.010. At relu4 the intervention transfers near ceiling (cross-train 0.913 joint, 0.911 separate; transfer 0.941 joint, 0.942 separate), and the largest joint-minus-separate difference across dimensions and metrics is 0.005.

These are dimension-16 means on the full sweep, which is unbalanced across dimensions: n falls from 135 joint, 129 intermediate, and 30 separate at dimension 1 to 68, 57, and 24 at dimension 16. On the balanced subset the same four values are 0.911, 0.911, 0.938, and 0.942.

A.4 Dimension Selection

We chose the subspace dimension from sweeps at conv3 and relu3. Joint and separate show no detectable difference at any dimension where the intervention localizes at all (Table 2). At conv3, transfer is near chance at dimension 1, from 0.53 to 0.58 across the three types, so no type comparison is informative there; it reaches 0.66 at dimension 4, and the types stay together from there on. At dimension 256 on the relu3 sweep the mean squared error configuration reaches only 0.531 cross-train, against 0.80 at dimension 64, and the types stay together there too (0.529 joint, 0.546 separate).

A.5 Non-linear DAS Architecture and Selection

The non-linear alignment map is a RevNet (Gomez et al., 2017) with B additive coupling blocks. Each block splits the flattened activation in half and updates the halves in turn, $y_1 = x_1 + F(x_2)$ then $y_2 = x_2 + G(y_1)$, where F and G are one-hidden-layer ReLU MLPs of width w. We run all four combinations of $B \in \{1, 4\}$ and $w \in \{32, 64\}$ at each of the twelve layer and dimension cells, relu2 at $d = 4$ and 16 and relu3 and relu4 at $d = 1$, 2, 4, 16, and 64, which is 48 runs in total. For each cell we report the configuration with the highest mean IIA across models. Training follows Section 3.2, taking the binary cross-entropy option and localizing shape alone. Each cell uses the same 78 models, and the non-linear runs use generated minimal pairs.

The linear map shows no detectable difference between the types at any layer or dimension, with AUCs between 0.42 and 0.53 (Table 2). Only the non-linear map separates them. It also localizes better, 0.942 against the rotation's 0.869, though not at every layer: at relu3 it reaches 0.852 at dimension 16, below the rotation's 0.878. Section 4.2 reports the relu4, dimension 1 cell, where separate leads with and AUC of 0.63.

A.6 Logit-Difference Readout

Reading the transfer interventions as logit differences rather than accuracies gives the same result: on circle and square pairs, under minimal and random sampling, joint and separate models do not

differ in the direction or magnitude of the induced logit change (Table 2; models with cross-train directional accuracy above 0.90).

B Additional Mechanistic Methods

B.1 Activation Steering

Activation steering pushes on the activations directly rather than swapping them. For an ordered pair of shapes, we take the difference between the average activation for one shape and the average for the other as a shape direction, add it to held-out images, and measure whether the output moves. We record the average change in the output logit and the fraction of images whose logit moves in the predicted direction. Steering uses the same cross-category pairs as DAS.

Steering shows a clear depth effect and no type difference (Table 2). The mean logit change grows from 3.1 at relu1 to 20.2 at relu4, and the fraction of images moving in the predicted direction is highest at relu3, at 0.528, 0.559, 0.751, and 0.658 across relu1 to relu4. The effect is consistent across shape pairs. Joint and separate overlap at every layer except relu1, where joint models move the output in the predicted direction more often (AUC 0.37, 95% CI 0.27 to 0.47, with separate as the positive class). On the balanced subset of 84 models, the interval covers chance (0.38, 95% CI 0.24 to 0.53), so we do not treat it as an effect.

B.2 Single Feature Map Patching

Single feature map patching asks whether one convolutional channel carries the category. We copy a channel's feature map from a source image into a base image, one channel at a time, and score the output against the counterfactual. We select the channel that works best on one shape pair, ellipse and capsule, and test whether it transfers to a different pair, circle and capsule.

Single-map transfer falls with depth, from 0.971 to 0.655 for joint models and from 0.957 to 0.612 for separate (Table 2). On the full population, joint models exceed separate at every depth, by 0.014 to 0.054. That is the predicted direction: the map is selected on ellipse to capsule, and tested on circle to capsule, and ellipse and circle are in the same category, so a joint detector, which uses one mechanism for both, should transfer. But the two types overlap at every depth (AUC 0.44 to 0.50 across conv1 to conv3), and on the balanced set the ordering reverses at conv3, where joint falls to 0.580 against 0.612 for separate. The stored results are constant within each convolutional block, so the eight layers carry three distinct values. These results are computed on 298 models: 135 joint, 30 separate, and 133 intermediate.

B.3 Probing

Probing reads the shape off the activations with a trained classifier. We train a linear probe and a one-hidden-layer MLP probe to predict the shape, a four-way target, and the category, a two-way target, from three views of a layer: the raw activations, the DAS subspace, and the part of the activations outside of it (the DAS complement). We use the controlled set, in which the same latents are rendered across all four shapes so that only the shape differs, and we split by template so the probe must generalize to unseen renderings rather than recognizing familiar ones. Probes are compared across five layers, and for the four-way shape target we separate errors within a category from errors across the category boundary. Probing compares joint against separate, with intermediate networks excluded.

Probing finds no detectable difference between the types (Table 2). Of the 90 cells (six probe variants, three targets, five layers), 88 fall within ± 2 SE of zero and 66 within ± 1 SE, and the largest standardized gap is 2.41. Joint and separate are indistinguishable at every layer and probe type, on the four-way shape target, the two-way category target, and within-category error.

The probes do show that shape is readable from the 16-dimensional DAS subspace by an MLP probe but far less well by a linear one (Table 2. The information may therefore be in the causal subspace in a form a linear read-out cannot recover.

B.4 Principal Component Analysis

PCA is the observational method. We project a layer's activations onto two or three principal components and ask whether the four shapes or the two categories form visible clusters, and whether joint and separate models differ. We use the controlled set: 100 templates, each rendered as all four shapes with the same object color, background color, and area, so the 400 images differ within a template only in shape. We ran PCA on the raw activations at conv3 and relu4 for 15 models, 5 of each type. This population predates the balanced set.

Figure 5 shows one joint and one separate model. The shapes separate at relu4 but not at conv3. Which component carries that separation differs by layer. At conv3 the circle separates from the ellipse along the third component in 11 of the 15 models and along the second in the other 4, and never along the first, where the separation is at most 0.11. At relu4 it separates along the second component in all 15. The figure plots the first two components.

Across all 15 models we computed one number: the distance between the ellipse and circle means on the first three components, divided by their average spread. At relu4, separate models keep the two shapes slightly further apart than joint models do, 0.674 against 0.635, with an AUC of 0.72 on 5 models per type whose ranges almost fully overlap. At conv3 the same measure is 0.349 for separate against 0.199 for joint, a wider relative gap but an AUC of 0.56, with a confidence interval from 0.16 to 1.00 at 5 models per type. Explained variance per component does not differ between the types at either layer.

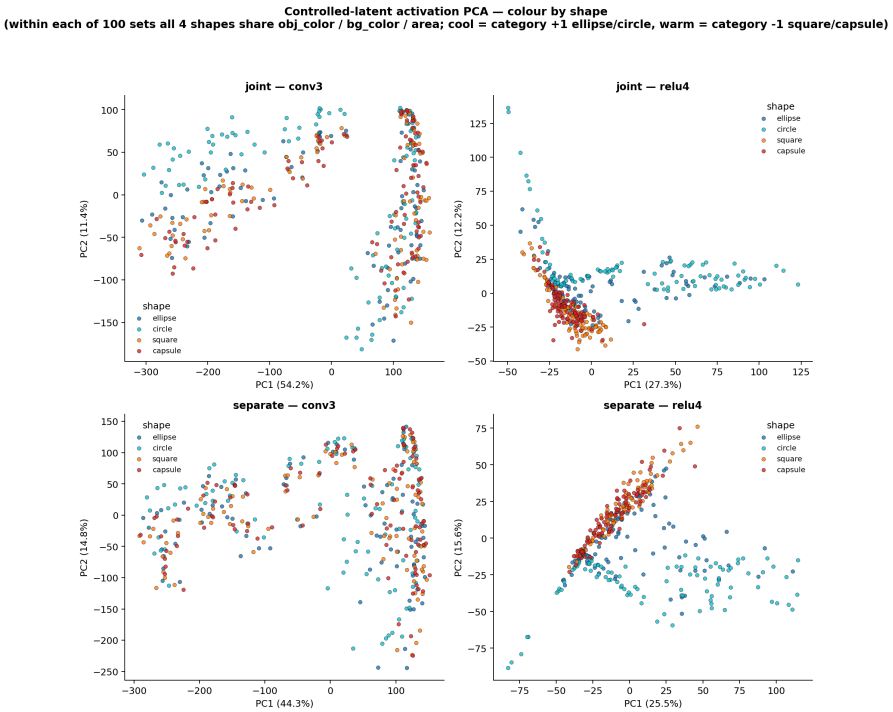

Figure 5: PCA of raw activations for one joint (top) and one separate (bottom) model at conv3 (left) and relu4 (right), on the 400 controlled images, colored by shape. Axes are the first two components with their explained variance.

C Non-linear DAS on the Arithmetic Task

We repeated the arithmetic localization with the non-linear map. Table 3 reports, for the rotation and for each RevNet configuration, the mean held-out R^2 for models labeled $-x - y$ and $-(x + y)$ at each layer, and the AUC for predicting the label with $-x - y$ as the positive class. These runs intervene after the ReLU and are a separate fit from the one behind Figure 3, which scores squared

error at the layer's linear output; the two agree at layer 1, 0.781 against 0.784, and not at layer 2, 0.680 against 0.791.

The RevNet raises R^2 at every layer and in every configuration. At layer 3 it rises from 0.62 under the rotation to 0.87 with one block and 0.94 with four, and depth adds more than width. The label signal does not survive. Under the rotation the $-x - y$ models are the better localized at all three layers, with AUC 0.78, 0.68, and 0.67. Under the RevNet ten of the twelve cells fall below 0.5, so the direction reverses rather than settling at chance, and no configuration holds one direction across the three layers.

Table 3: Non-linear DAS on the arithmetic task: mean held-out R^2 by transfer label and the AUC for predicting the label, by alignment map and layer. AUC takes $-x-y$ as the positive class. $n = 18$ $-x - y$, 17 $-(x + y)$, from 100 trained models. R^2 is scored on 1000 counterfactual pairs drawn independently of the 100,000 used for fitting; the two draws are independent samples rather than a partition.

Map	Layer	R^2, $-x - y$	R^2, $-(x + y)$	AUC
Rotation	1	0.895	0.808	0.781
	2	0.668	0.587	0.680
	3	0.651	0.579	0.670
RevNet B=1, w=32	1	0.959	0.977	0.320
	2	0.754	0.746	0.526
	3	0.819	0.887	0.363
RevNet B=1, w=64	1	0.966	0.979	0.337
	2	0.770	0.759	0.536
	3	0.838	0.894	0.408
RevNet B=4, w=32	1	0.998	0.999	0.683
	2	0.925	0.970	0.356
	3	0.933	0.961	0.392
RevNet B=4, w=64	1	0.998	1.000	0.585
	2	0.934	0.968	0.376
	3	0.934	0.961	0.379