DECOMPOSING DECOMPOSITION: WHEN AND HOW DO MODELS LEARN COMBINATORIAL STRUCTURE?

Anonymous authors
Paper under double-blind review

ABSTRACT

There is now abundant evidence that neural networks learn to decompose solutions to large problems by learning smaller subroutines for subproblems, which can often be reused and recombined, sometimes even facilitating helpful forms of compositional generalization. Comparatively little is known, however, about what drives this decomposition, and where we should expect to find it. The present work confronts this question by studying the dynamics of learning various *combinators*, that is, higher-order functions that determine how simpler functions combine. We first study which combinators can be learned by Transformer-based language models across model sizes and data configurations. We then find that the model's generalization accuracy on novel combinations is strongly correlated with its successful realization of the task's intermediate causal variables. We explain how the model identifies and learns these intermediate variables by tracing the gradients of training examples that share the same intermediate value yet have different downstream functions. We find that these gradients drive the clustering of the intermediate variable representations, and such clustering leads to both the emergence of the causal variables identified by prior interpretability research, and the model's compositional generalization capacity.

1 INTRODUCTION AND MOTIVATION

Common wisdom about natural and artificial learning systems is that they must break down larger problems into smaller, more tractable, potentially reusable, subproblems (Simon, 1962). A language model, for example, whose ultimate task is to predict language tokens, must learn to perform a number of subsidiary tasks along the way: parsing, semantic role labeling, co-reference resolution, and so on (Tenney et al., 2019). Larger tasks are often decomposable into relatively modular subtasks.

Much of interpretability research is premised on a literal reading of a nearby modularity claim: the model itself must carry out the various subtasks in such a way that the respective computational pathways can be understood and studied in relative isolation from each other. We assume it makes sense to ask circumscribed questions like, "How does this model perform addition?," or "How do transformers recognize co-referents?," without needing to address every other thing a model is doing simultaneously. A divide-and-conquer strategy, at the level of tasks, is assumed possible.

A (usually tacit) justification for this assumption is that representational decomposition may be part of the explanation for how a model learns complex tasks in the first place. In order to solve intricate tasks that have some inherent combinatorial structure, a model needs to respect that combinatorial structure in its computations, for example, by representing the results of intermediate computations for subtasks. This assumption has been abundantly validated in the literature (Merullo et al., 2024; Lindsey et al., 2025). Indeed, every successful mechanistic analysis of model computation can be seen as further confirmation of the assumption, insofar as the analysis locates "intermediate" variables that play a causal mediating role in a meaningful (sub)task (Geiger et al., 2025; Mueller et al., 2026). Furthermore, detection of the relevant mechanisms has been shown to be a reliable guide to (certain forms of compositional) generalization (Huang et al., 2025; Wu et al., 2025).

One special case of these phenomena has been explored in the literatures on *multi-hop* and (closely related) *implicit* reasoning (Biran et al., 2024; Wang et al., 2024; Yao et al., 2025; Ye et al., 2026). This line of research investigates how models answer questions like, "What is the capital of the country where Stonehenge is located?" Because a model is unlikely to store this piece of information

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.

explicitly, it must first retrieve the relevant country, and only then retrieve the capital of that country. Existing work has begun exploring the conditions under which models succeed at tasks like these.

Despite the abundance of evidence that neural learning models do decompose problems—often in predictable ways following the "true" decomposition of the generating process—comparatively less is known about precisely when, why, and how this emerges during training. The present work targets these questions by looking at learning trajectories for various *combinators*, that is, higher-order functions that determine how functions and arguments are combined.

Taking as a typical example combinator B (from Curry 1930)—the one implicated in most multi-hop reasoning settings—we conduct a thorough analysis of the mechanisms induced by a range of models that successfully learn it, and elucidate the precise learning pressures that lead to emergence of the requisite *intermediate variables*. For instance, what drives a model to represent "England" as an intermediate step in the example above?

By characterizing the training examples that push a model in the direction of encoding such variables, we are able to predict when and how a model will succeed. Compared to previous work, the present analysis goes deeper into the mechanisms and training dynamics behind decomposition. But it also goes wider by putting the combinatorial structure itself into the spotlight. We study a range of combinators other than B, using our analysis to explain why some combinators are harder to learn than others, and even to predict transfer patterns between related combinators. The result is a more detailed and comprehensive understanding of when and how models induce combinatorial structure.

2 Combinators and Learnability

The study of combinators goes back 100 years (Schönfinkel, 1924). Using λ-notation, a very simple example is Curry's B, which takes two functions, f and g, and returns the composition $g \circ f$:

$$\mathsf{B} : \lambda fg.\, \lambda x.\, g(f(x))$$

This is merely the tip of a combinatorial iceberg. Here are more examples (also illustrated in Figure 8 in Appendix A.3):

$$\begin{aligned}
\mathsf{C}_1 &: \lambda fh.\, \lambda xy.\, h(f(x,y)) & \mathsf{C}_6 &: \lambda fgh.\, \lambda xyv.\, h(f(x,y), g(v,y))\\
\mathsf{C}_2 &: \lambda fgh.\, \lambda xy.\, h(f(x), g(y)) & \mathsf{C}_7 &: \lambda fg.\, \lambda xyvu.\, g(f(x,y), f(v,u))\\
\mathsf{C}_3 &: \lambda fgh.\, \lambda xyvu.\, h(f(x,y), g(v,u)) & \mathsf{C}_8 &: \lambda fgh.\, \lambda xyvu.\, h(g(x,y), f(v,u))\\
\mathsf{C}_4 &: \lambda fg.\, \lambda xyv.\, g(f(x,y), f(v,y)) & \mathsf{C}_9 &: \lambda fgh.\, \lambda xyvu.\, h(u, g(v, f(x,y)))\\
\mathsf{C}_5 &: \lambda fgh.\, \lambda xy.\, h(f(x,y), g(x,y)) & \mathsf{C}_{10} &: \lambda fgh.\, \lambda xyvu.\, h(g(f(x,y), v), u).
\end{aligned}$$

Combinators like these are widely assumed to ground core linguistic capacities, such as syntactic parsing (see Steedman 2000) and semantic evaluation (see, e.g., Donatelli & Koller 2023). And many are explicitly attested in reasoning patterns for even very simple neural models. For instance, $\mathsf{C}_2(\mathbf{1}_{=})$ is the "hierarchical equality" circuit from Geiger et al. (2024), while $\mathsf{C}_4(\mathbf{1}_{\leq}, \wedge)$ is the "price-tag" algorithm from Wu et al. (2023). And it is easy to find more complex examples. Somehow, models have effectively learned to apply combinators like these flexibly to wide ranges of arguments. The fundamental question motivating our work is why and when we should expect this.

2.1 (When) Do Transformer Models Learn Combinators?

To explore the question systematically, we shall first gather a series of empirical results on small Transformer models that we can understand reasonably well. Under what conditions—that is, with what data, what sizes of models, etc.—do models learn the combinators listed above?

Data and Model. For binary-function combinators, let $\mathcal{V}$ be a set of V values and $\mathcal{F}$ a pool of N atomic functions. Each $f \in \mathcal{F}$ is a *balanced random mapping* $f : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$, with each output value occurring equally often over all input pairs. For example, C_3 computes $h(f(x,y), g(v,u))$ for $f, g, h \in \mathcal{F}$ and $x, y, v, u \in \mathcal{V}$. For B, which composes two unary functions, we instead vary the sizes of the input ($\mathcal{X}$), intermediate ($\mathcal{V}$), and output ($\mathcal{Y}$) spaces independently. For $x \in \mathcal{X}$, the

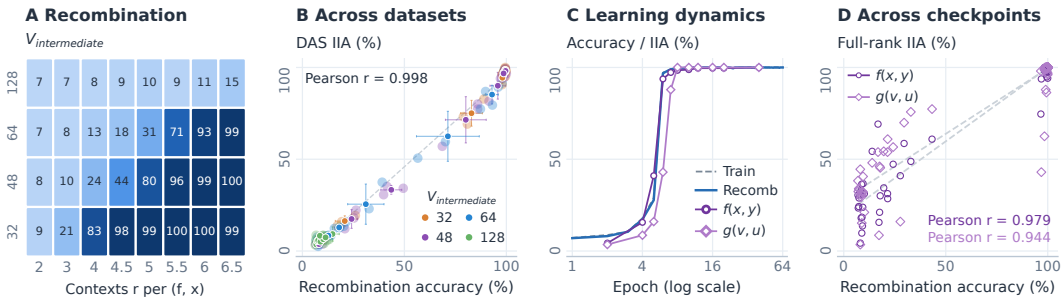


Figure 1: **Recombination accuracy correlates with localization of intermediate variables. (A)** Recombination accuracy for B, varying the number of intermediate values and outer functions seen per (f, x), with two layers, and hidden width 128. **(B)** DAS IIA for $f(x)$ versus recombination accuracy across these datasets; points and error bars show means and standard deviations over three seeds. **(C)** Training accuracy, recombination accuracy, and full-rank IIA for $f(x, y)$ and $g(v, u)$ during training on C_3, with $N = 4$, $V = 16$, two layers, and hidden width 256. **(D)** Full-rank IIA versus recombination accuracy across checkpoints from 15 C_3 models.

answer is $g(f(x))$, where $f : \mathcal{X} \to \mathcal{V}$ and $g : \mathcal{V} \to \mathcal{Y}$ are balanced random mappings drawn from separate pools $\mathcal{F}$ and $\mathcal{G}$, respectively. We randomly sample training instances such that every atomic input-output mapping is used at least once in computing the answer to training instances. We train relative position decoder-only Transformers from random initialization, varying depth and hidden width, using AdamW and no weight decay. Loss is only computed on the final answer token.

Evaluation. We primarily evaluate whether the model learns to succeed at *recombination*: whether it can solve unseen combinations of atomic functions and input arguments when all needed atomic mappings are covered in training. This includes *novel function combinations*, where the tested composition of atomic functions has never appeared in training, but each has been seen together with other functions.

Generalization results. We find that models can achieve high recombination accuracy on B, C_3, C_9, C_8, C_6, C_7, and C_{10}. In particular, models' generalization seems sensitive to the number of downstream functions and to how many inputs map to the same intermediate value (Figure 1A and Appendix B.1). This appears robust to model architecture (Appendix B.3) and weight decay can improve reuse but it is not necessary (Appendix B.4). We also find that these models generalize while they can just memorize the training set (Appendix B.5). Remarkably, they can learn to recombine even without explicit labels for the atomic functions in the prompts (Appendix B.6).

For C_5 and C_4, by contrast, models can be successfully trained to minimize training set loss, but they do not achieve high recombination accuracy, thus showing no evidence of generalization.

2.2 Generalization Correlates with Presence of Causal Variables

We then ask whether the model solves the task by learning the task's ground-truth compositional structure, that is, whether they compute the intermediate variables—for example, $f(x, y)$ and $g(v, u)$ in C_3 and $f(x)$ in B. Using interchange interventions (Geiger et al., 2021; 2025) and distributed alignment search (DAS; Geiger et al. 2024), we find that we can always successfully localize the intermediate variables in models with sufficiently high recombination accuracy (see Figure 20 for localization details). We observe a strong correlation between interchange intervention accuracy (IIA) and models' recombination accuracy, both across datasets (Figure 1 A,B) and across checkpoints (Figure 1 C,D). The same pattern holds for models trained on inputs without explicit function labels (Appendix C.3.2). These findings extend the work of Huang et al. (2025) to compositional generalization in a synthetic setting.

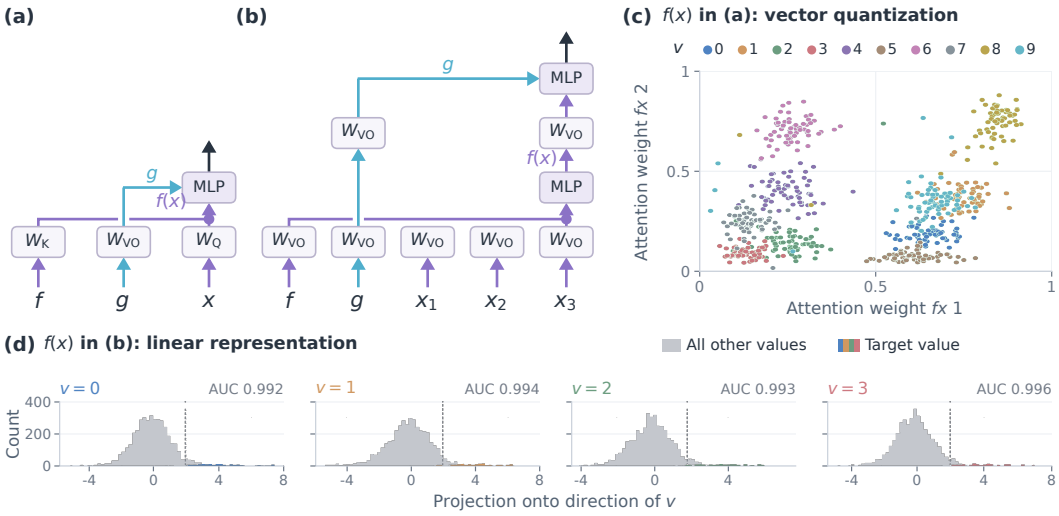


Figure 2: **Mechanism for computing the intermediate variable $f(x)$ and $f(x_1, x_2, x_3)$ in combinator B. (a)** For single-token argument, the intermediate variable $f(x)$ is computed at the attention weight. **(b)** For multi-token argument, the intermediate variable is computed at the first layer MLP. **(c)** For single-token argument, $f(x)$ is represented by its nearest centroid. **(d)** For multi-token argument, $f(x)$ is represented linearly with one value per direction.

3 Explaining the Emergence of Causal Variables

We begin by analyzing the learning dynamics of the simplest combinator B. This is a good test case because it involves only a single intermediate variable, namely the output of function f. Intuitively, encoding this intermediate result is necessary in order to exhibit the behaviors canvassed above in Section 2.1, an intuition further supported by the results in Section 2.2. We then move on to study the more complex C_3, which seems to require encoding two intermediate variables: the outputs of (binary functions) f and g, respectively.

3.1 Combinator B

We investigate combinator B with variants of f taking in a single token argument x or a multi-token argument $x_1, \ldots, x_n$ for $n = 2, 3$. As in the previous section, we focus on the same small Transformer models with 1-2 layers.

3.1.1 Mechanisms for routing arguments and intermediate variables

We start by analyzing the algorithm implemented by a model that has successfully learned the compositional structure, i.e., allowing novel combination of f and g.

Localization of variables. We first analyze where the arguments and functions can be localized using DAS, as illustrated in Figure 2 (a) and (b). In the single-token argument case, x is routed through an attention query, f is routed through an attention key, while g is routed through an attention value. $f(x)$ is causally encoded in the attention weights and then passed to the attention output at the last token. Critically, $f(x)$ is only encoded in the attention weight, but not in the attention value. In fact, $f(x)$ in attention value causes generalization failures (see Appendix B.2). Then, g is applied to the intermediate variable at the MLP. In the multi-token argument case, v is not computed until the first layer MLP. The second layer attention then gathers g, which is applied to v in the second layer MLP.

Representation of the intermediate variable v. Now we zoom into the intermediate variable and analyze how its value is computed. For the single-token argument case, the attention weight between f and x naturally captures a bilinear interaction: $a_{f,x} = \sigma(W_k E_f \cdot W_q E_x)$, where W_k, W_q

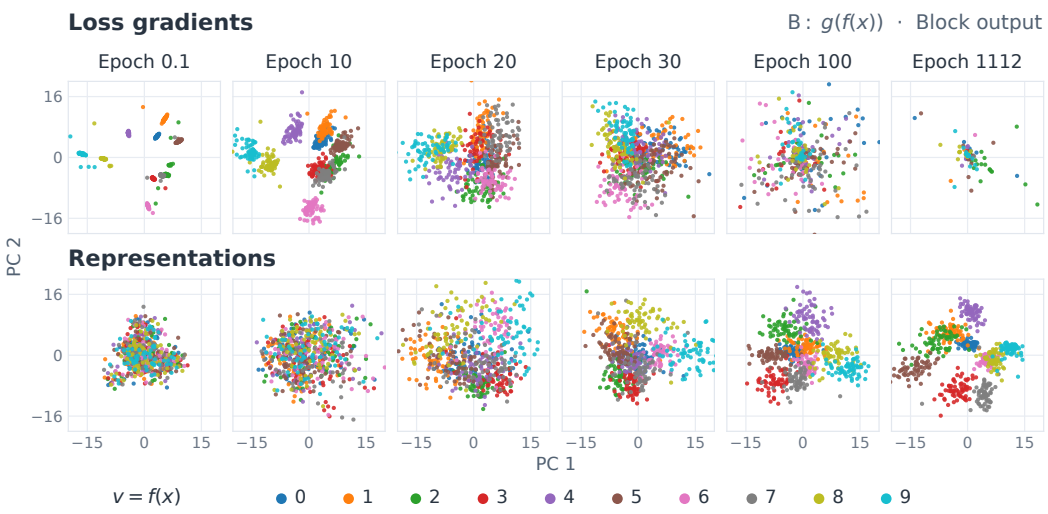


Figure 3: **Gradients and representations during learning for B: $\lambda f g.\, \lambda x.\, g(f(x))$.** PCA projections of loss gradients **(top)** and representations **(bottom)** at the block output of a one-layer model trained on B. Each point represents a fixed (f, x) pair, averaged over outer functions g and colored by the intermediate value $f(x)$. We use the same examples, correctly answered by the final model, at every checkpoint. Gradients cluster by $f(x)$ early in training, before clear clusters appear in the representations. PCA is fitted separately in each panel.

are the weights for attention key and query; E_f, E_x are the embeddings of f and x plus position embedding and normalization; and σ is the softmax. Given a Transformer with H attention heads, $f(x)$ is encoded in an h-dimensional code, $h \leq H$, with one $a_{f,x}$ from each attention head. Each $a_{f,x}$ value is roughly clustered into l buckets, which allows at most l^h different codes, each of which corresponds to a different value of $f(x)$, as shown in Figure 2 (c). The representation of this simple intermediate variable v surprisingly presents a counter-example to the strong interpretation of the Linear Representation Hypothesis (LRH) (Elhage et al., 2022; Park et al., 2023): $f(x)$ is not encoded one value per direction, i.e., one cannot linearly separate one value of $f(x)$ from the others. After $f(x)$ is computed in the attention weights, f and x are largely suppressed by the attention value weights, keeping only a shared representation of f and x. For the multi-token argument case, the $x_1, \ldots, x_n$ are first merged into an argument representation in the attention output. The computation of $f(x_1, \ldots, x_n)$ happens in the first layer MLP, similar to the widely studied factual knowledge retrieval mechanism (Geva et al., 2021). In the MLP output, values $f(x)$ are linearly represented, i.e., each direction represents a value, as shown in Figure 2 (d). We provide more supporting evidence in Appendix C.

3.1.2 Which Examples Identify the Intermediate Variable?

Since the label y does not directly encode any information about the intermediate variable $f(x)$, the question naturally arises: Where does the learning signal of the intermediate variable $f(x)$ come from? In this section, we show that the learning signal comes from symmetries in the data: The set of examples with the same $f(x)$ value always maps to the same output y regardless of the choice of g. This set of examples identifies $f(x)$.

Let $z \in \mathbb{R}^{|V|}$ be the logits of model $\mathcal{M}$, $p = \sigma(z) \in [0, 1]^{|V|}$ be the predicted probability, and y be the one-hot encoded label, where V is the output vocab and σ is the softmax function. It is easy to check that the gradient of cross entropy loss $\mathcal{L}$ is:

$$\frac{\partial \mathcal{L}}{\partial z} = p - y \tag{1}$$

Lemma 1 (Identical aggregate gradients) *Let $S_{f,x} = \{(f, g_i, x) | g_i \in \mathcal{G}\}$ be the set of all training examples that agree on f and x (but possibly vary on g_i). The aggregate gradient on this set of*

examples—i.e., the sum of gradients in equation 1 over $S_{f,x}$—is

$$\Delta z_{f,x} = \underbrace{\sum_{g_i \in S_{f,x}} p_i}_{\textit{Predictions}} - \underbrace{\sum_{g_i \in S_{f,x}} y_i}_{\textit{Labels } Y_{S_{f,x}}}, \tag{2}$$

The second term $Y_{S_{f,x}}$ is an aggregated label histogram, where $y_i = \mathbf{e}_{g_i(f(x))}$, and $\mathbf{e}_i$ is a unit vector with only the ith entry being 1. For any two sets S_{f_1,x_1} and S_{f_2,x_2}, where $f_1(x_1) = f_2(x_2) = v$, we are guaranteed that $Y_{S_{f_1,x_1}} = Y_{S_{f_2,x_2}}$. In other words, the set of examples with the same intermediate value $v = f(x)$ share the very same learning signal.

Intuitively, aggregating over f, x "compresses" the input space into a small label space, with multiple examples explained by the same "compression." Aggregating over g, x, or f, g, by contrast, fails to have this property: It results in the same number of distinct label histograms as the number of unique g, x or f, g pairs. In this sense, the gradient in Lemma 1 provides a *contracting force* on the representation of $f(x)$, in the sense that examples with the same intermediate value $f(x)$ will be clustered together in the internal representations.

Theorem 1 (Identifiability of $f(x)$) *By Lemma 1, two intermediate values v, v' are indistinguishable by the aggregated gradient $\Delta z_{f,x}$ if and only if they share the same label set Y_S; in other words, iff $g_i(v) = g_i(v')$ for all $g_i \in \mathcal{G}$. With a sufficiently large number of g's, the label set can uniquely identify the intermediate variable v with high probability.*

Specifically, we can offer the following probabilistic argument: Let $a = f_1(x_1)$ and $b = f_2(x_2)$, and let $\pi_{\text{eq}} = P(a = b)$ be the "prior" probability these two values are equal. In our setup, $\pi_{\text{eq}} = 1/m$, where $m = |\mathcal{V}|$. Furthermore, let $q = |\mathcal{Y}|$ be the size of the output space. Consider some list $g_1, \ldots, g_k : \mathcal{V} \to \mathcal{Y}$ of random maps, and let $E_k = \bigcap_{j=1}^{k} \{g_j(a) = g_j(b)\}$. Then the "posterior" probability of $a = b$ after witnessing E_k is:

$$\begin{aligned} P(a = b \mid E_k) &= \frac{\pi_{\text{eq}}}{\pi_{\text{eq}} + (1 - \pi_{\text{eq}})q^{-k}} \\ &= \frac{1}{1 + (m-1)q^{-k}}. \end{aligned}$$

In other words, the probability that they are *not* equal becomes vanishingly small with increasingly many g_i's. This already helps explain the behavior in Figure 1, where increasing the number k of g_i's leads to better generalization accuracy.

Empirical validation. We verify that the learning signal derived in Lemma 1 is present in actual training. We measure the gradient direction and the representation at the final layer residual stream across model checkpoints, as shown in Figure 3. The gradient over representation of the intermediate variable is clustered by its value in the first 20 epochs, which coincide with the phase where the intermediate variable is formed and the model starts to generalize (shown in Appendix Figure 13). At the same time, the representations gradually form clusters based on the value of the intermediate variable. As the cluster forms, the gradient signal becomes weaker, because the optimization pressure has largely been absorbed by the representation.

3.2 Combinator C_3

As a reasonably representative example of one of the more complex combinators, consider C_3, which, recall, takes seven arguments, f, g, h, x, y, v, u, and returns $h(f(x, y), g(v, u))$.

3.2.1 Mechanisms for routing arguments and intermediate variables

We analyze a two-layer model that has learned the compositional structure, i.e., recombines familiar f, g and h on unseen inputs. C_3 has two intermediate variables, $f(x, y)$ and $g(v, u)$, each computed from a two-token argument; the same analysis applies to both. We provide the supporting evidence in Appendix C.

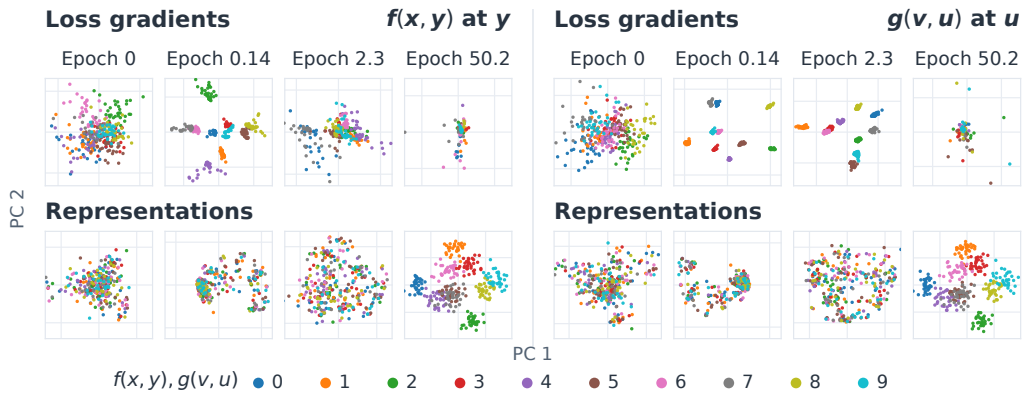


Figure 4: **Gradients and representations during learning for C_3.** Plotting conventions follow Figures 3; measurements are taken at the first-block output.

Localization of variables. Under causal attention, y is the first position that sees f, x and y, and u is the first position that sees g, v and u; the two intermediate variables are computed at these two positions (Figure 20 in Appendix C). In the first layer, the last argument y is routed through the attention query, while f and x are routed through the attention values; y also enters the MLP directly through the residual stream. The attention output at y thus encodes f, x and y, but not yet $f(x, y)$. The value $f(x, y)$ is first computed in the first-layer MLP at y, and $g(v, u)$ in the same MLP at u. The second-layer attention then gathers h, z_L and z_R at the last token, and h is applied to the two values in the second-layer MLP.

Representation of the intermediate variables. The attention at y merges f, x and y into an argument representation, and $f(x, y)$ is computed in the first-layer MLP. Inside the MLP, the value first becomes linearly separable at the gated product, $\mathrm{SiLU}(W_{\mathrm{gate}}r) \odot W_{\mathrm{up}}r$ with r the MLP input, although there it occupies directions that carry little of the variance; the down projection then writes it into the residual stream, where the sixteen values form clusters and $f(x, y)$ is linearly represented, one value per direction. The same MLP weights compute $g(v, u)$ at u. In the second layer, attention at the last token collects h, $f(x, y)$ and $g(v, u)$. The outer function is applied to the two values in the second-layer MLP, as g is applied to $f(x)$ in B.

3.2.2 Which Examples Identify the Intermediate Variable?

The argument of Section 3.1.2 applies to each intermediate variable of C_3: all examples with the same value of $f(x, y)$ map to the same answer under the same h and the same $g(v, u)$, so the pair $(h, g(v, u))$ plays the role that g plays in identifying $f(x)$. As shown in Appendix B.1, models' generalization on C_3, just as on B, is also sensitive to the number of downstream functions and to how many inputs map to the same intermediate value. For both $f(x, y)$ and $g(v, u)$, we find evidence similar to B (Figure 4): the loss gradients at the first layer block output cluster by the intermediate value early in training, while the representations form clusters only late in training.

3.3 Generalizing Beyond B and C_3

In the previous two sections we offered fairly detailed analyses of combinators B and C_3, but the general ideas apply more broadly. As a first window into this generality, we note the empirical facts that both the behavioral (Appendix B.1) and representational (Appendix C.3.1) patterns we have observed in B and C_3 extend to other combinators, such as C_9, C_2, and C_1.

Since we cannot give a gradient-level analysis that applies simultaneously to all combinators and all model families, it will be helpful to elevate to the more abstract level of the probabilistic argument given above in Section 3.1.2. Suppose we have some putative intermediate variable V: this could be $f(x)$ in B, or $g(v, u)$ in C_3, or even $g(v, f(x, y))$ in C_9. For each such variable V, we can consider its *continuation*, $\lambda U.\, U(V)$, within the larger expression: in B the continuation of $V = f(x)$ is $\lambda g.\, g(V)$, while in C_3 it is $\lambda gh.\, \lambda vu.\, h(V, g(v, u))$, and so on.

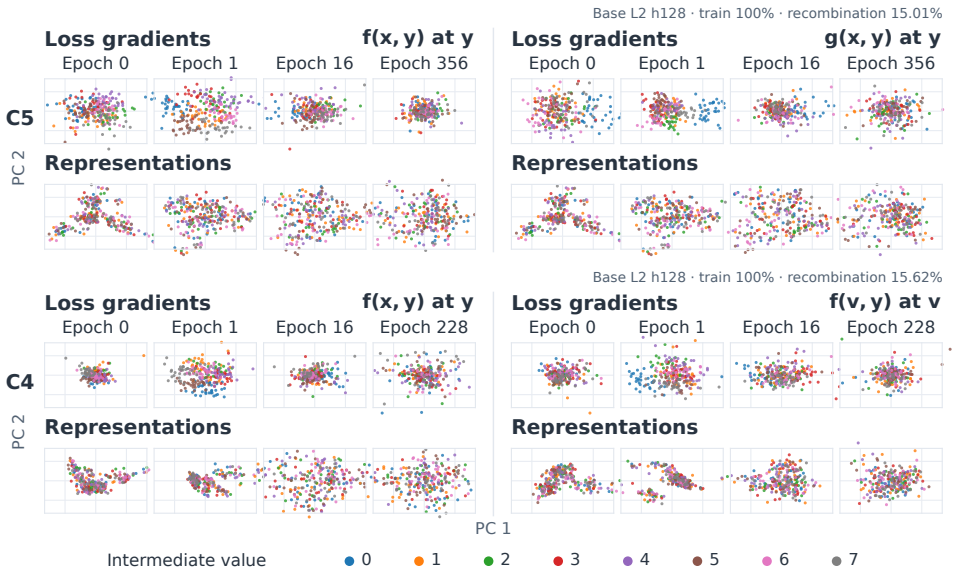


Figure 5: **Gradients and representations during learning for C_5 and C_4.** Plotting conventions follow Figures 3–4; measurements are taken at the first-block output. Early gradients show some intermediate-value structure, but final representations do not form clear value clusters. Both models achieve perfect training accuracy but generalize poorly.

The basic shape of the probabilistic argument in Sections 3.1 and 3.2 is that the variable V will be induced, roughly speaking, just when there are contexts U to witness different values of V with the same value of U. As established, for example, in Theorem 1, relatively generic learning pressures will push the same values of V together, resulting in a "subspace" that encodes this intermediate variable V. Moreover, as also established earlier in Section 2.2, the emergence of the relevant intermediate variables is evidently responsible for some of the most interesting generalization abilities, viz. recombination accuracy. The model will generalize only when those variables are induced.

Because the same basic argument should apply equally well to C_9, C_8, C_2, and C_1, this helps to explain why we see similar success with those combinators. It may also help elucidate why we see failures with other combinators. Consider, for instance, combinator C_5. The potential intermediate variable is $V = f(x, y)$. However, because the values x, y are "tied" between f and g, there is no simple continuation whose arguments we can count. A similar issue arises with C_4, where both f and y are shared between the variable and its continuation. While we have not shown that there is no other way for a model to learn those intermediate variables, our results point strongly in the direction of that conclusion: the intermediate values are not in fact encoded, and even after training to convergence the models do not generalize in the relevant ways. The gradient patterns that we found for the other combinators fail to hold for C_5 and C_4; see Figure 5.

Finally, it is worth mentioning that our analysis also makes predictions about potential *transfer* between related combinators. This is important because, in more naturalistic settings, it is plausible that many combinators must be learned simultaneously, and having learned some combinators may make it easier to learn others. Without pursuing this direction too deeply, we note that models can reuse atomic functions *across* combinators in certain settings. We jointly train on two combinators that share the same pool of N atomic functions. In one combinator, we hold out half of the functions from role h during training and evaluate them in that role at test, while these held-out functions are available in the other combinator. Transfer from C8 to C3 achieves high test accuracy across datasets.

4 Related Work

Connections to implicit reasoning. The combinator B studied in this work is a more general form of the implicit reasoning tasks studied in the literature (Press et al., 2023; Biran et al., 2024; Wang et al., 2024; Yang et al., 2025; Yao et al., 2025; Ye et al., 2026; Chang et al., 2026). Implicit reasoning

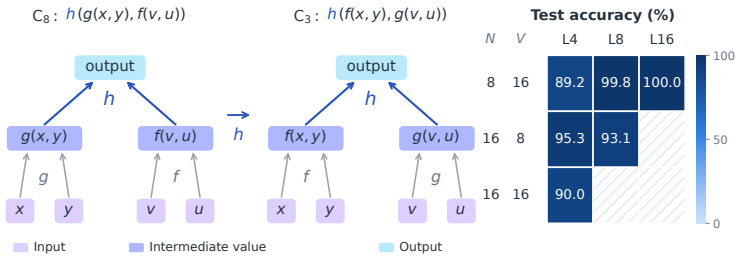


Figure 6: **Reuse atomic functions at role h across combinators.** (Left) Causal graph for C8 and C3. (Right) The transfer accuracy of funtions at role h.

tasks often share the form "(entity 1, relation 1, relation 2) $\mapsto$ entity 3", where the two relations are connected through an intermediate entity, e.g., "the capital of the country that San Francisco is in" infers an intermediate entity "United States". Prior work has causally localized representations of the intermediate entity in pre-trained LLMs (Biran et al., 2024; Yang et al., 2024; Yao et al., 2025) and noticed that clustering of representations of intermediate entity coincides with the "reasoning phase", where the model can generalize to unseen relation compositions (Ye et al., 2026). Our findings in Section 2.2 generalize the causal variable accounts to a broader set of combinators, and our explanations in Section 3.1 connect these two accounts, showing that both the causal variable and the cluster of representations can be explained by the optimization pressure in training data that identifies the intermediate variable.

Explaining data properties required for learning combinatorial structures. Recent work has attempted to provide a data-centric account of generalization behaviors on algorithmic tasks that often involve combinatorial structures. For example, Liu et al. (2022) identifies a critical data size for grokking on modular addition task. Wang et al. (2024) further narrows it down to inferred/atomic facts ratio for implicit reasoning tasks, which is in fact correlated with the data generation parameters studied in this work, e.g., $|\mathcal{V}|$ and $|\mathcal{G}|$. Deceasing $|\mathcal{V}|$ or increasing $|\mathcal{G}|$ allows more function reuse and hence increases the inferred/atomic facts ratio. Chang et al. (2026) formalizes a similar coverage concept through functional equivalence, which is the property that leads to identical aggregated label histograms for examples sharing the same intermediate value in Section 3.1.2. We take a step further and connect this data property to the representation structures of the intermediate variable observed in prior work through gradient signals. Overall, our findings align with the critical data properties identified in prior work, and we further bridge these data properties with model representations through gradients, explaining how these data properties are learned by the model.

5 Conclusion and Looking Ahead

The aim of this paper has been to explore when, why, and how neural models decompose complex problems into simpler subproblems. Combinators offer a powerful lens on this problem, as they offer a precise and general regimentation of compositional structure. Starting with the simple combinator B, and then moving on to more complex combinators like C_3, we have isolated some key properties that facilitate successful learning, tracing these all the way to features of the gradient that push models toward clustering intermediate values together, ultimately leading to the emergence of intermediate causal variables that support complex counterfactual predictions.

While the present study already takes us beyond what was previously understood in the literature, it is still only a first step in a more comprehensive analysis. A few areas that we believe merit more investigation include: Other kinds of learning signals, e.g., transitivity studied in Hwang & Park (2026); Transfers between combinators, especially in curriculum learning settings; Localization of combinators; Explaining more natural data, such as natural language. Zooming out further, the need to understand precisely how models generalize is becoming ever more poignant, witness the recent increase in "misaligned" models that appear to generalize in surprising ways (Betley et al., 2025). Combinatorial generalization—for instance, the recombination patterns studied here—is closely tied to what intermediate variables a model induces over the course of training. This brings together mechanistic interpretability with the study of training dynamics and properties of training data. A more comprehensive understanding will need to unify all of these; we hope this paper marks an important step in that direction.

AI USE STATEMENT

In this work, we used generative AI tools for research execution, i.e., generating training and plotting scripts, and drafting sections of the appendix, i.e., summarizing results.

We have not used generative AI tools for aiding writing (except for spell and grammar checking), retrieval and discovery, or research ideation. Proving mathematical claims and generating synthetic datasets are not applicable to this work. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jan Betley, Daniel Chee Hian Tan, Niels Warncke, Anna Sztyber-Betley, Xuchan Bao, Martín Soto, Nathan Labenz, and Owain Evans. Emergent misalignment: Narrow finetuning can produce broadly misaligned LLMs. 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. 4043–4068. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/betley25a.html.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14113–14130, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.781. URL https://aclanthology.org/2024.emnlp-main.781/.

Hoyeon Chang, Jinho Park, Hanseul Cho, Sohee Yang, Miyoung Ko, Hyeonbin Hwang, Seungpil Won, Dohaeng Lee, Youbin Ahn, and Minjoon Seo. Characterizing pattern matching and its limits on compositional task structures. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VCjlm003WL.

Haskell B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52 (3):509–536, 1930. doi: 10.2307/2370619.

Lucia Donatelli and Alexander Koller. Compositionality in computational linguistics. *Annual Review of Linguistics*, 9(Volume 9, 2023):463–481, 2023.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022. URL https://arxiv.org/abs/2209.10652.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9574–9586, 2021.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In *Proceedings of the 3rd Conference on Causal Learning and Reasoning (CLeaR)*, pp. 160–187, 2024.

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

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, Online and Punta Cana, Dominican Republic, November

2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL https://aclanthology.org/2021.emnlp-main.446/.

Jing Huang, Junyi Tao, Thomas Icard, Diyi Yang, and Christopher Potts. Internal causal mechanisms robustly predict language model out-of-distribution behaviors. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Ofa1cspTrv.

Hyeonbin Hwang and Yeachan Park. Intrinsic task symmetry drives generalization in algorithmic tasks, 2026. URL https://arxiv.org/abs/2603.01968.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=6at6rB3IZm.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. Circuit component reuse across tasks in transformer language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fpoAYV6Wsk.

Aaron Mueller, Jannik Brinkmann, Millicent Li, Samuel Marks, Koyena Pal, Nikhil Prakash, Can Rager, Aruna Sankaranarayanan, Arnab Sen Sharma, Jiuding Sun, Eric Todd, David Bau, and Yonatan Belinkov. The quest for the right mediator: Surveying mechanistic interpretability for NLP through the lens of causal mediation analysis. *Computational Linguistics*, 52(1):331–378, March 2026. doi: 10.1162/coli.a.572. URL https://aclanthology.org/2026.cl-1.10/.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Causal Representation Learning Workshop at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=T0PoOJg8cK.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92 (3–4):305–316, 1924. doi: 10.1007/BF01448013. URL https://doi.org.

Herbert A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P19-1452.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokking of implicit reasoning in transformers: A mechanistic journey to the edge of generalization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=D4QgSWxiOb.

Yiwei Wu, Atticus Geiger, and Raphaël Millière. How do transformers learn variable binding in symbolic programs? 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. 67284–67299. PMLR, 2025.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah Goodman. Interpretability at scale: Identifying causal mechanisms in Alpaca. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning? In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10210–10229, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.550. URL https://aclanthology.org/2024.acl-long.550/.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3971–3992, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.205. URL https://aclanthology.org/2025.findings-acl.205/.

Yuekun Yao, Yupei Du, Dawei Zhu, Michael Hahn, and Alexander Koller. Language models can learn implicit multi-hop reasoning, but only if they have lots of training data. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9684–9702, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.490. URL https://aclanthology.org/2025.emnlp-main.490/.

Jiaran Ye, Zijun Yao, Zhidian Huang, Liangming Pan, Jinxin Liu, Yushi Bai, Amy Xin, Liu Weichuan, Xiaoyin Che, Lei Hou, and Juanzi Li. How do transformers learn implicit reasoning? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=19ygs48nOa.

A Experimental Setup

A.1 Types of generalization evaluated

In this work, we mainly consider two levels of generalization (illustrated in Figure 7):

- **Recombination.** We test whether models can solve unseen combinations of atomic functions and input arguments when all required atomic mappings are covered in training. This includes new input combinations for a composition of functions already seen during training. For example, for C_3, a test example can use the same (f, g, h) as training examples but a different combination of input arguments (x, y, v, u). We sample training examples such that every atomic mapping is used in computing at least one training answer, and exclude training examples from the test set. Thus, the model must recombine mappings seen in training to answer an unseen example.
- **Novel function combination.** We further test recombination when the combination of atomic functions itself has never appeared in training. For C_3, we hold out selected combinations (f, g, h) and exclude all examples using these combinations from training. Each individual function is still seen in the same role with other functions, and all required atomic mappings are covered in training. At test time, the model must therefore combine functions that it has seen separately but never together in this composition.

There are two additional types of reuse we have not studied in this paper and are left for future work:

- **Input holdout.** To test whether models can reuse atomic functions across roles, we hold out half of the inputs to the function at a certain position in the combinator. Consider C_3, which computes $h(f(x, y), g(v, u))$, and split the V^2 combinations of input arguments (x, y) into two equally sized sets. Every atomic function is seen in role f during training, but only with inputs from the first set; for testing, we evaluate on inputs from the second set. This split is the same for all atomic functions. We sample training examples such that all atomic mappings are fully covered across roles. Thus, the test requires the model to apply a function to inputs seen with that function in other roles, but never in role f.
- **Function holdout.** We further test whether models can reuse a function in a role where it has never been seen during training. We split the N atomic functions into two equally sized sets. For the same C_3 example, for training, f is drawn only from the first set, while g and h are drawn from the pool of all N functions; at test time, f is drawn from the second set. All atomic mappings are covered in training through other roles. In contrast to held-out inputs, where each function is seen in role f with other inputs, the model must now apply a function in that role without seeing any training examples of that function in that role.

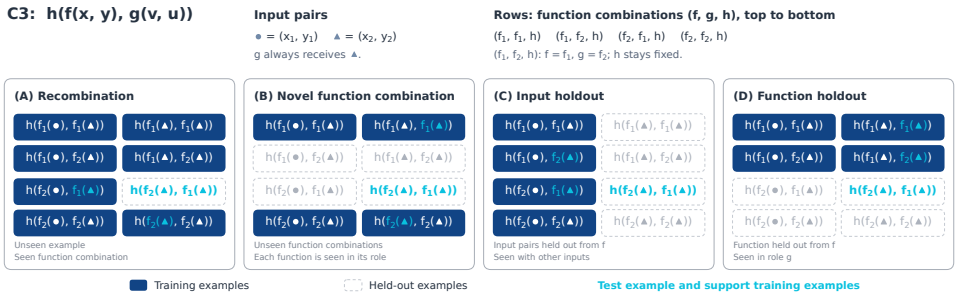


Figure 7: **Four generalization tests illustrated with C3, $h(f(x, y), g(v, u))$.** Each cell represents one example in the generated datasets for each combinator. Blue cells are training examples; dashed borders mark held-out examples; blue-green text shows the test example and parts of training examples that support the corresponding recombination. In this illustration, h stays fixed and h always receives (x_2, y_2) as input. We use circles and triangles to represent two distinct input pairs (x_1, y_1) and (x_2, y_2), respectively.

A.2 EXAMPLE FORMAT

A prompt specifies the combinator, its atomic functions, and its input arguments. No intermediate results are provided as targets. For example, C_3 maps (f, g, h, x, y, v, u) to $h(f(x, y), g(v, u))$. The prompt format is `C3 f_i f_j f_k x y v u`, followed by one answer token. Suppose $f_0(1, 2) = 3$, $f_1(3, 0) = 1$, and $f_2(3, 1) = 2$. The prompt `C3 f_0 f_1 f_2 1 2 3 0` evaluates the two inner calls to 3 and 1, then outputs $f_2(3, 1) = 2$.

A.3 COMBINATORS

Figure 8 shows the causal graph of all the combinators used in the experiment.

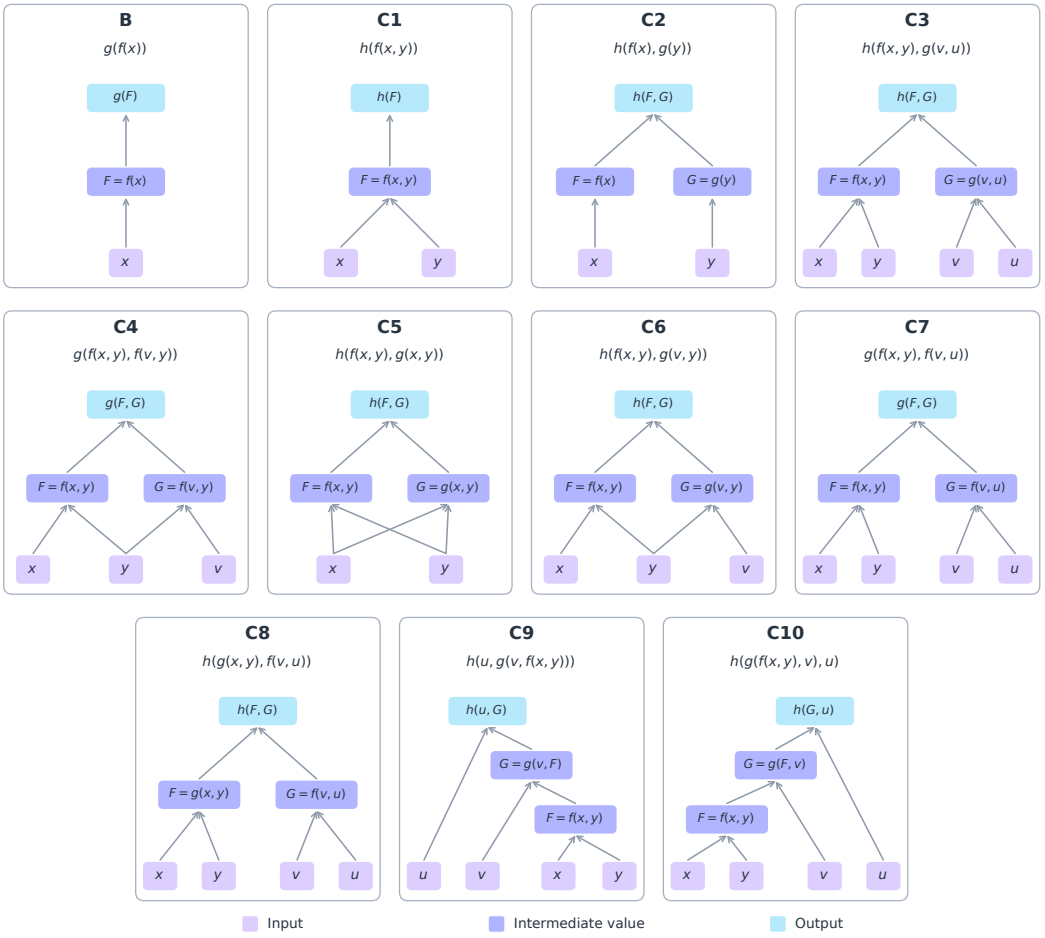


Figure 8: **Causal graph of combinators we use in experiments.**

B Additional Behavioral Results

B.1 Full generalization results

We examine how compositional generalization varies with the combinator (Figure 8), the number of atomic functions N, the value-domain size V, and model size. We only report results for models trained to convergence and fit the training data perfectly.

Recombination results are shown in Figure 9 .

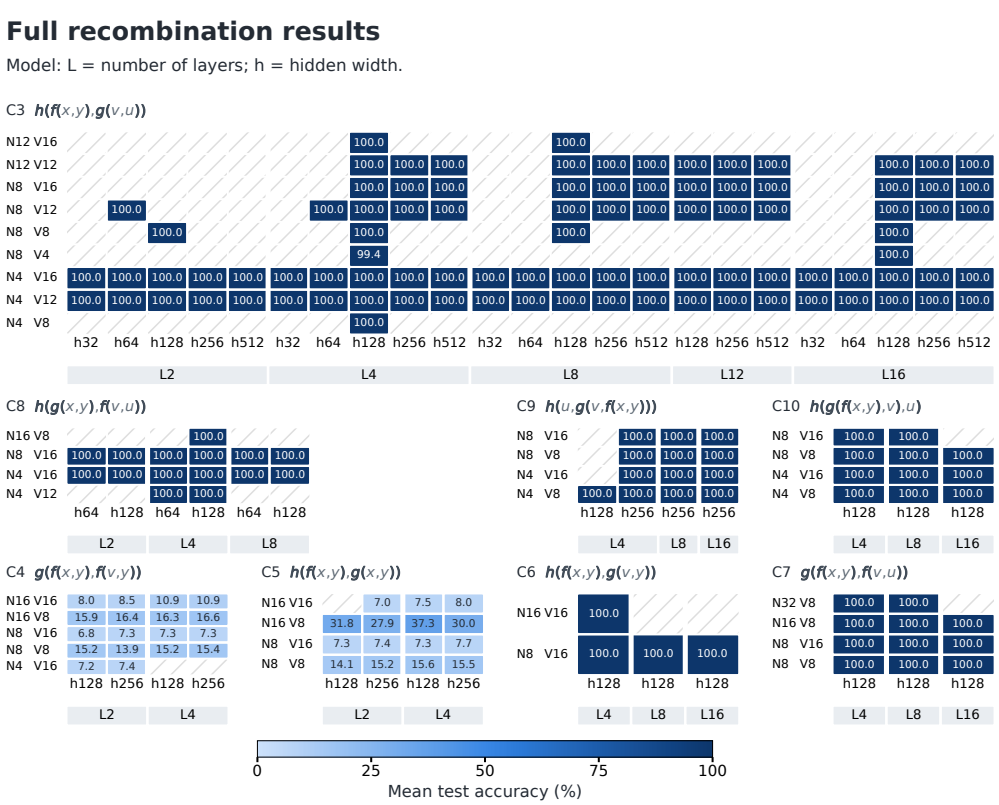


Figure 9: **Full recombination generalization results.** Each panel shows a binary-function combinator. Rows show (N, V); columns show hidden width h, grouped by depth L. Cells show mean recombination accuracy over seeds. Since the input spaces could be large, we by default downsample the datasets to 8,192, 4,096, and 2,048 examples per composition at $N = 4, 8, 12$, respectively.

For B, we further vary the training data and model size to examine when a model that fits the training set generalizes (Figure 10). For $g(f(x))$, inner functions map inputs to intermediate values, and outer functions map intermediate values to answers. We independently vary the number of inner functions, outer functions, inputs, intermediate values, and outputs. With a two-layer OLMo2 model of hidden width 128, both accuracies generally increase as each exact (f, x) call appears with more downstream functions g (panel A). Increasing context coverage also increases the number of training examples in this setting. Model width is another factor that improves generalization substantially. For example, with 256 input values, 64 intermediate values and 6.5 contexts, widening a one-layer model from 32 to 128 raises novel function-pair accuracy from 12.6% to 98.9% (panel B). Model depth does not consistently help.

For C_3, we also vary the number of intermediate values and the number of downstream contexts per exact inner call (Figure 11). In the original setting, a single N controls the function pools and a single V controls all variable cardinalities. Here, we specify the inner and outer function-pool sizes and the input, intermediate, and output cardinalities separately. This allows us to change the intermediate-value size while keeping the function pools and input and output sizes fixed. With



Figure 10: **Data and model variation for** $\mathsf{B} = g(f(x))$**.** (A) A two-layer OLMo2 model with hidden width 128 is trained with 256 possible input values while varying the number of possible intermediate values $f(x)$ and the mean number r of distinct downstream functions g per exact (f, x) call. (B) The same eight datasets are each used to train models with different numbers of layers L and hidden widths h. Rows are sorted first by r, then by input cardinality, then by intermediate cardinality. All settings use eight inner functions, eight outer functions and 16 possible output values. The size of the training dataset is $8V_{\text{input}}r$. Colored cells show mean accuracy (%) over initialization seeds 0, 42 and 123; the data seed is fixed at 0. We only report models with mean training cross-entropy at most 10^{-4}.

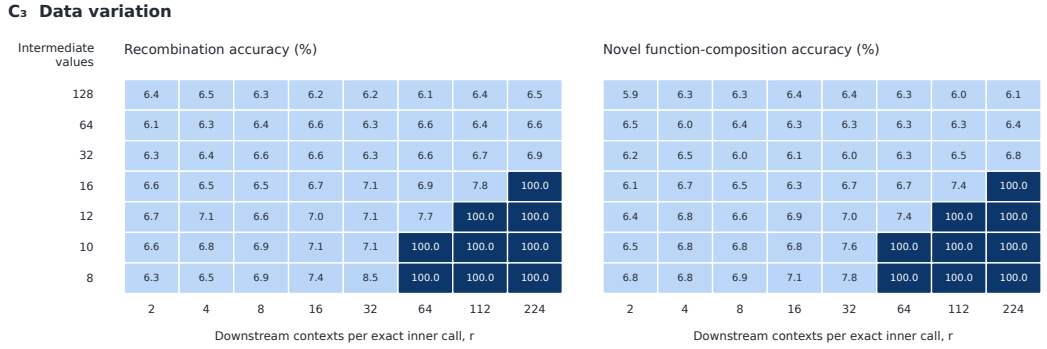


Figure 11: **Data variation for** C_3**.** We train a two-layer OLMo2 model with hidden width 128 while varying the number of possible intermediate values and the number r of distinct downstream contexts per exact inner call. All settings use a shared pool of eight inner functions for f, g, a separate pool of eight outer functions for h, 16 possible values for each raw input, and 16 possible output values. For a fixed call (f, x, y), a downstream context specifies (h, g, v, u); for a fixed call (g, v, u), it specifies (h, f, x, y). Every call appears in exactly r contexts, giving $2{,}048r$ training examples. Left: recombination accuracy on new input combinations within observed function compositions. Right: novel function-composition accuracy on held-out combinations of h with the inner-function pair. Cells show mean accuracy (%) over initialization seeds 0, 42, and 123; the data seed is fixed at 0. We only report results of models achieve 100% training accuracy and have a mean training cross-entropy at most 10^{-4}.

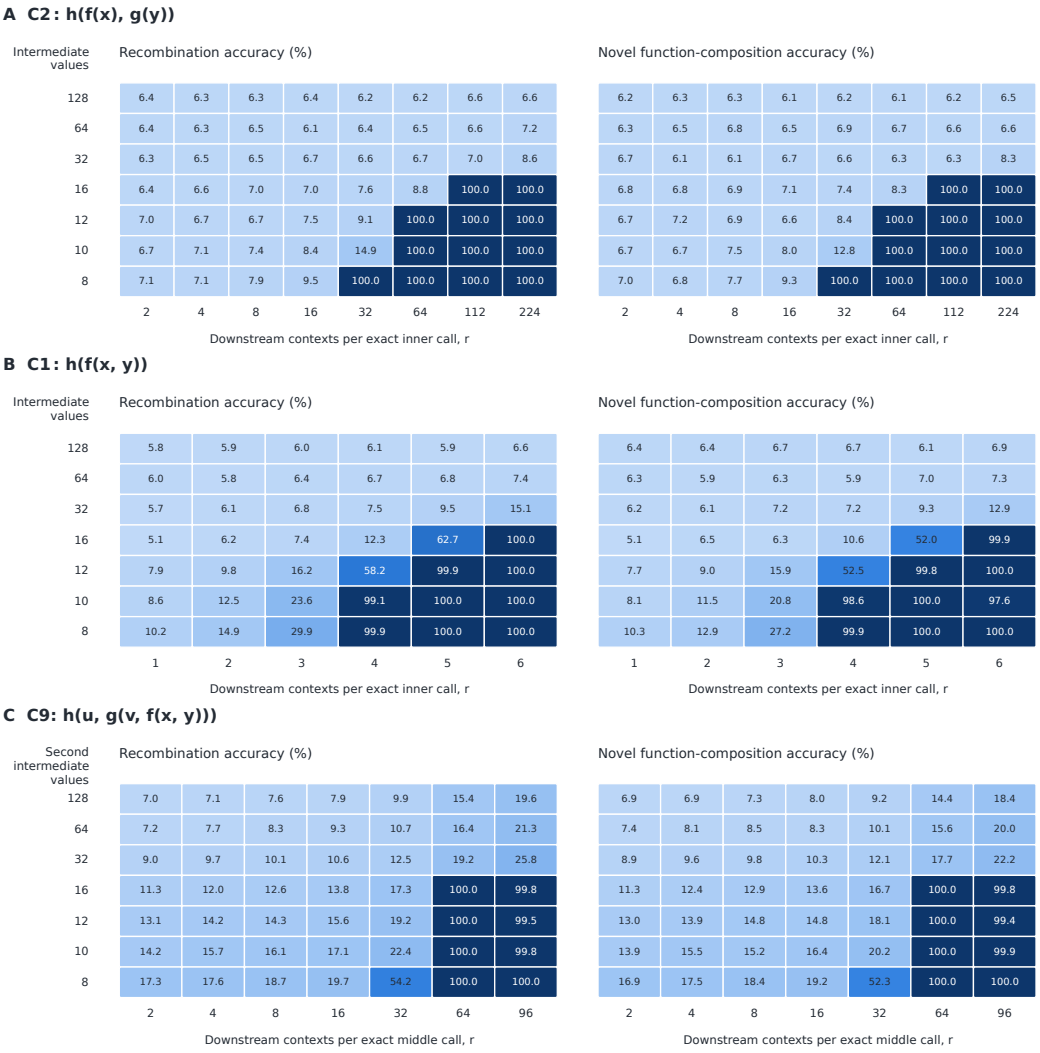


Figure 12: **Data variation for more combinators.** The setting is the same as above. Two-layer base OLMo2 models with hidden width 128 use eight inner functions, eight outer functions, and 16 output values. For C2, contexts are distinct (h, g, y) combinations for a fixed (f, x) call, with the corresponding definition for the other branch; for C1, they are distinct h functions for a fixed (f, x, y) call. For C9, contexts are distinct (h, u) combinations for a fixed (f, g, x, y, v) call; each (f, x, y) call appears with two fixed (g, v) contexts. Training-set size increases with r: $2{,}048r$ examples for C2 and C1, and $4{,}096r$ for C9.

these controls separated, we observe behavioral patterns similar to B: smaller intermediate spaces reach high generalization accuracy with fewer training contexts.

As shown in Figure 12, repeating these sweeps for $\mathsf{C}_2 = h(f(x), g(y))$ and $\mathsf{C}_1 = h(f(x, y))$ reveals the same overall pattern: smaller intermediate spaces reach high recombination accuracy with fewer training contexts.

B.2 Weight Initialization

While data properties are the dominant factors determining model generalization behaviors, for models trained on datasets with weak learning signals, e.g., the anti-diagonal entries in Figure 1 A, different weight initializations could also lead to high variance in generalization accuracy. Here we focus on analyzing a set of 32 runs that are trained on the same dataset with the same ordering of

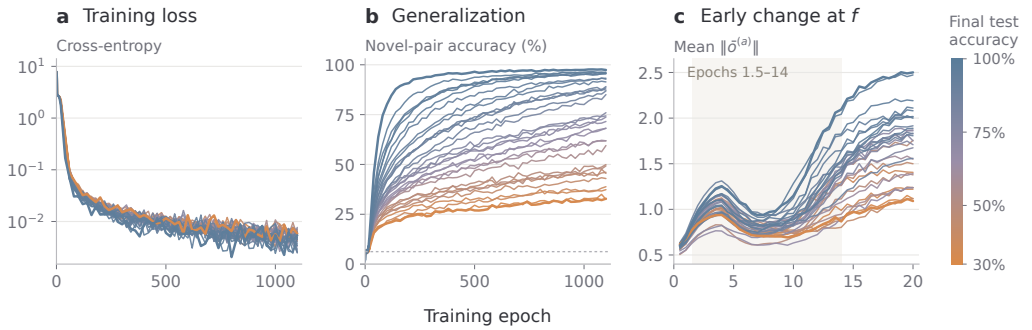


Figure 13: **With weak signals in training data, weight initialization could also induce variance in generalization accuracy.** The plots show a set of 32 runs with different random weight initializations, but trained on the same data with the same order. Color encodes the generalization accuracy. (a): All runs reach a close to zero training loss. (b): Generalization accuracy on novel function combinations diverges, ranging from 30% to 99%. (c): Generalization accuracy can be predicted purely from model weights that are related to how f token write to attention outputs from early on in training.

training examples but different weight initializations. All runs have close to 0 training loss, but their test accuracies span from 30% to 99%.

Building on the mechanisms identified in Section 3.1, we show that we can predict the final generalization accuracy from the very first few epochs in training. The key to generalization is whether the model can develop a strong signal of $f(x)$ early on in the attention output, which requires the attention value to be in a direction that is invariant to f, as otherwise f will also get encoded in the attention output. This implementation depends on the fixed direction attending to f writes, i.e., $\bar{o}^h = W_O^h \, (W_V^\top \bar{e})_h$, where h is the index to the attention head and $\bar{e}$ is the f embeddings. Note that $\bar{o}^h$ only depends on model weights and is independent of model inputs. In Figure 13 c, we show that $\bar{o}^h$ is predictive of model generalization accuracy.

B.3 Other architectures

We also examine generalization across several model architectures. We find that good recombination generalization on combinator B is possible with several model architectures. We compare OLMo2 and GPT-2 Transformers, LSTM, and MLP with GELU activations. All models are trained from scratch to predict $g(f(x))$ from (f, g, x), with supervision only on the final answer. The LSTM reads the three input embeddings as a sequence; the MLP concatenates them and applies two hidden layers followed by an output layer.

Figure 14 shows how the results depend on the data and training settings. The two Transformer grids vary the number of possible intermediate values and the number of downstream functions paired with each inner-function call (f, x). Both architectures generally perform better with fewer intermediate values and more downstream contexts. In the LSTM and MLP grids, we vary model size, learning rate, weight decay, and training budget on a fixed dataset. We find that their results also depend strongly on these choices. Figure 15 examines individual model and training configurations.

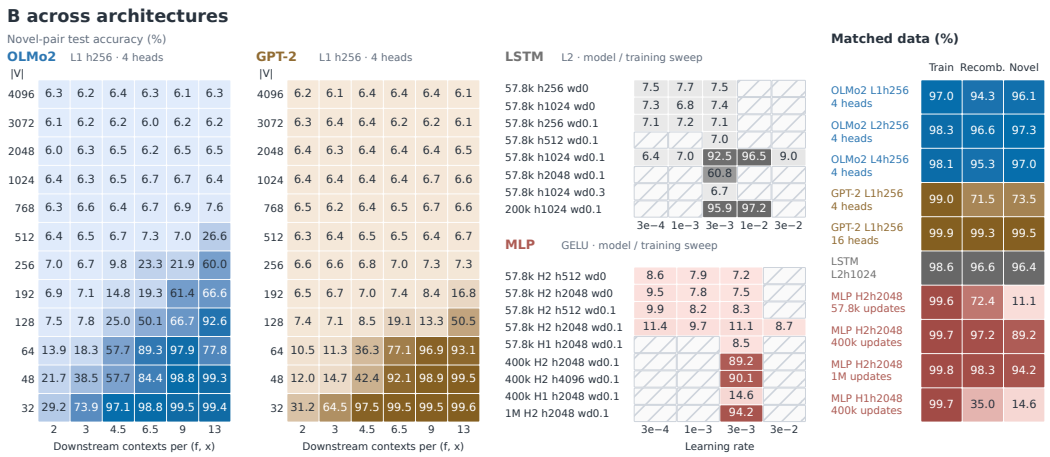


Figure 14: **Generalization on** B **across architectures.** Cells show accuracy in percent; colors identify architectures, with darker cells indicating higher accuracy. Rows give the number of possible intermediate values, $|V|$; columns give the mean number of distinct downstream functions g paired with each (f, x) in training. The OLMo2 and GPT-2 grids use one layer, hidden width 256, and four attention heads. Each grid summarizes 376 data configurations in 72 cells, averaging available seeds within each configuration and then averaging configurations within each cell. The LSTM and MLP grids show 19 and 18 individual run endpoints on a fixed task with eight inner functions, eight outer functions, 4,096 inputs, 256 intermediate values, and 16 possible answers. Here L denotes layer count, H denotes MLP hidden-layer count, h denotes hidden width, wd denotes weight decay, and 57.8k denotes 57,824 updates. Hatched cells have no experimental result. The rightmost table uses the same fixed task and compares training accuracy, accuracy on unseen combinations, and accuracy on novel function compositions. Uniform chance accuracy is 6.25%.

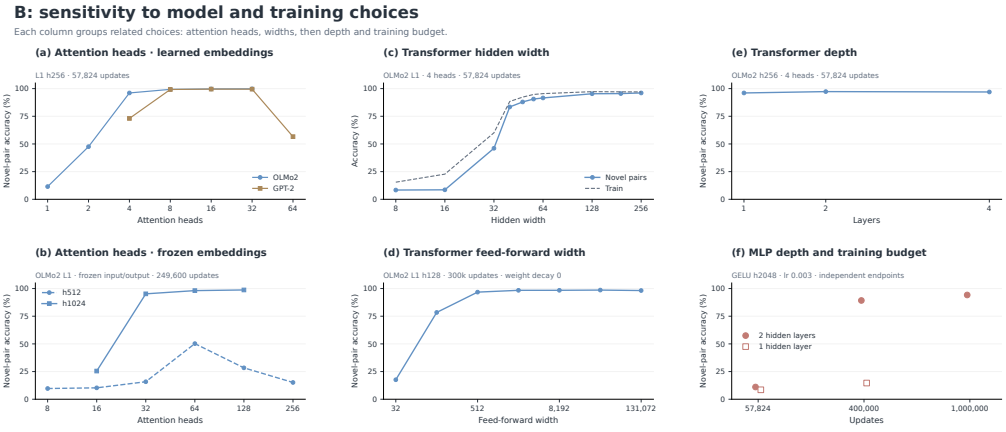


Figure 15: **Generalization on** B **across architectures and training configurations.** All panels use the fixed task described in Figure 14, with 6.5 downstream contexts per (f, x) on average. Values are novel function composition accuracy in percent, averaged over seeds. (a) Attention-head count for OLMo2 and GPT-2 with learned embeddings; hidden width and parameter count remain fixed within each architecture. (b) Attention-head count for OLMo2 with frozen input and output embeddings, at hidden widths 512 and 1,024. (c) OLMo2 hidden width; the dashed line additionally shows training accuracy. (d) OLMo2 feed-forward width. (e) OLMo2 layer count. (f) Training budgets for GELU MLPs with one or two hidden layers of width 2,048.

B.4 WEIGHT DECAY

We find that weight decay can improve reuse, but is not necessary for high input-holdout accuracy in these models. This aligns with recent findings from Hwang & Park (2026), which challenges the necessary role of weight decay reported in prior work. We compare C3 models trained with weight decay 0 and 0.1 at $N = 4$, $V = 12$, and hidden width 128 (Figure 16). Under input holdout, the four-, eight-, and sixteen-layer models already achieve 95.82%, 96.55%, and 92.41% accuracy without weight decay. Weight decay raises these values to 99.77%, 99.58%, and 99.03%; the two-layer model improves from 14.62% to 99.22%. Under function holdout, the eight-layer model improves from 37.71% to 58.87%.

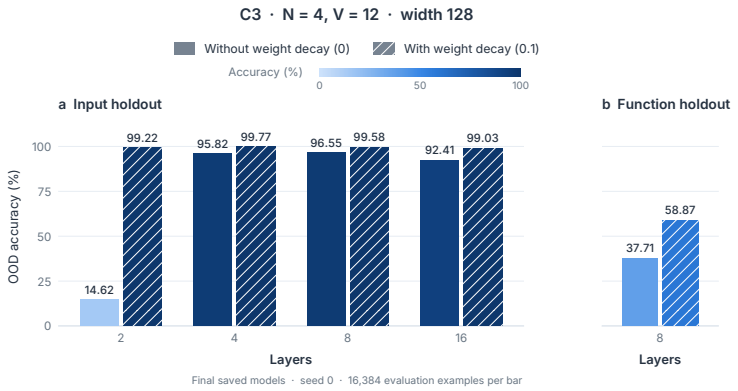


Figure 16: **Generalization with versus without weight decay.**

B.5 MODEL HAS SUFFICIENT CAPACITY TO IMPLEMENT A MEMORIZATION SOLUTION

One explanation for generalization is that the model cannot store the training examples individually. To test this explanation at a setting where models generalizes in C3, we train the same L2/h128 architecture on 6,560 distinct random prompts, matching prompt length and the token alphabet at each position. Each prompt receives an independently sampled output label that remains fixed throughout training. We match seed-specific initialization, objective, optimizer, and maximum training horizon to the structured task. All three random-token models fit their training sets, while all three structured models fit training and achieve 100% recombination accuracy (Figure 17). See results across more model sizes in Figure 18.

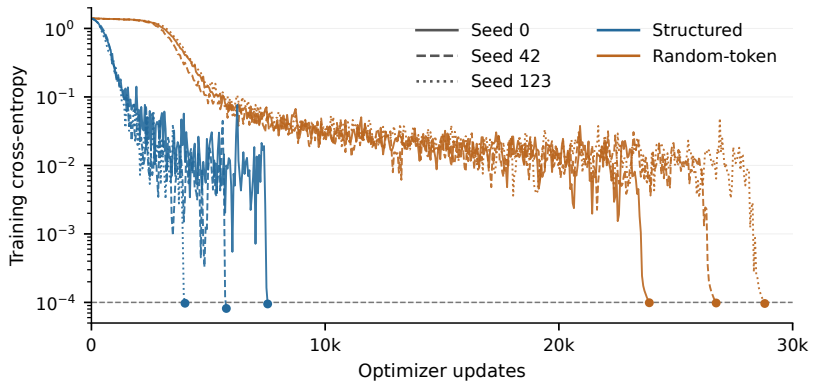


Figure 17: **Models can implement both the memorization solution and the generalization solution.** Recorded full-training census losses for models trained on the three structured and three random-token C3 datasets at $N = V = 4$, L2/h128 and 6,560 examples. The horizontal line is the 10^{-4} loss threshold; dots mark first qualifying endpoints.

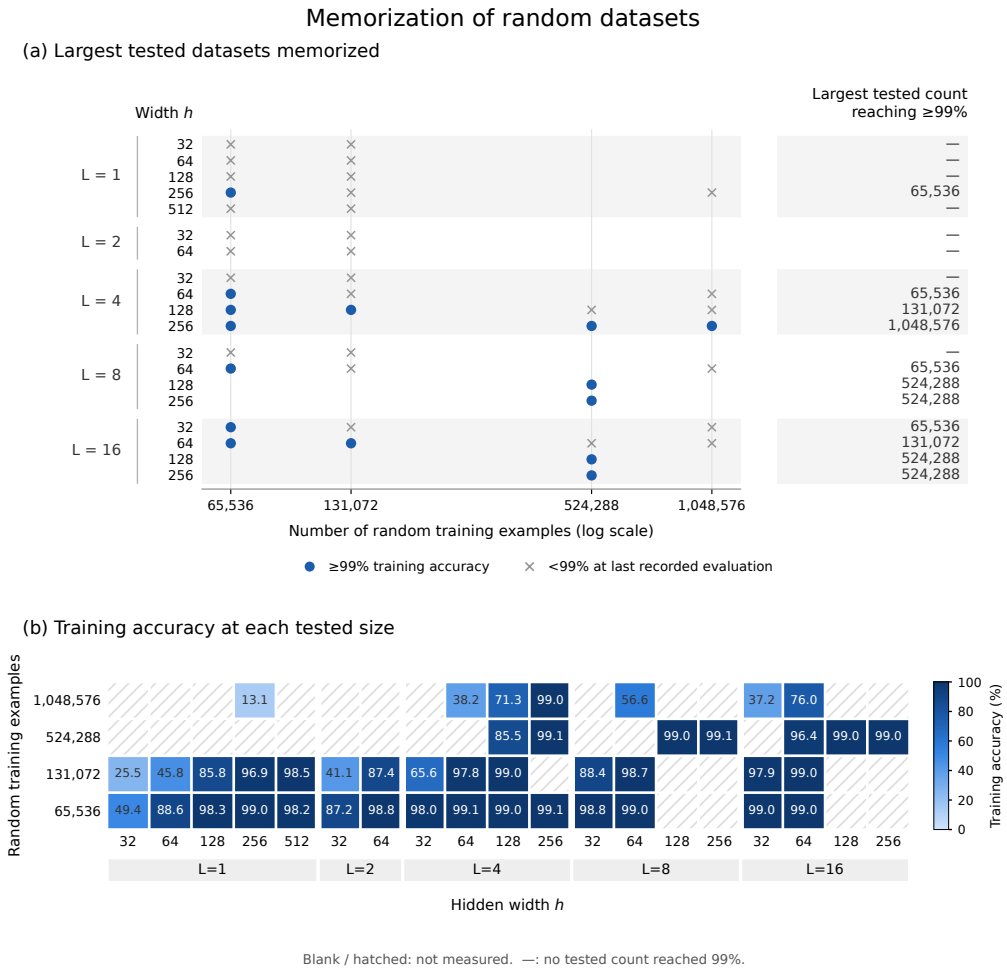


Figure 18: **Memorization of random datasets.** Both panels use the same models and datasets. **(a)** Circles mark training accuracy of at least 99%; crosses mark lower accuracy. The right column shows the largest tested number of examples reaching 99%; a dash means no tested count reached this criterion. **(b)** Cells show the best measured training accuracy (%) at each model size and number of examples.

B.6 Explicit versus implicit composition identifiers

Explicit prompts specify each atomic function, which may help a model separate the component computations. To test whether decomposition also emerges without these identities, we replace `C3 f_i f_j f_k x y v u` with `g_i x y v u`, where the single opaque token `g_i` identifies the entire composition. Both formats supervise only the final answer. We find that the model can still learn to decompose and generalize with this implicit input format, although it learns more slowly. As expected, once the model generalizes, we can successfully localize the intermediate variables (Figure 19). Localization across checkpoints are shown in Figure 25.

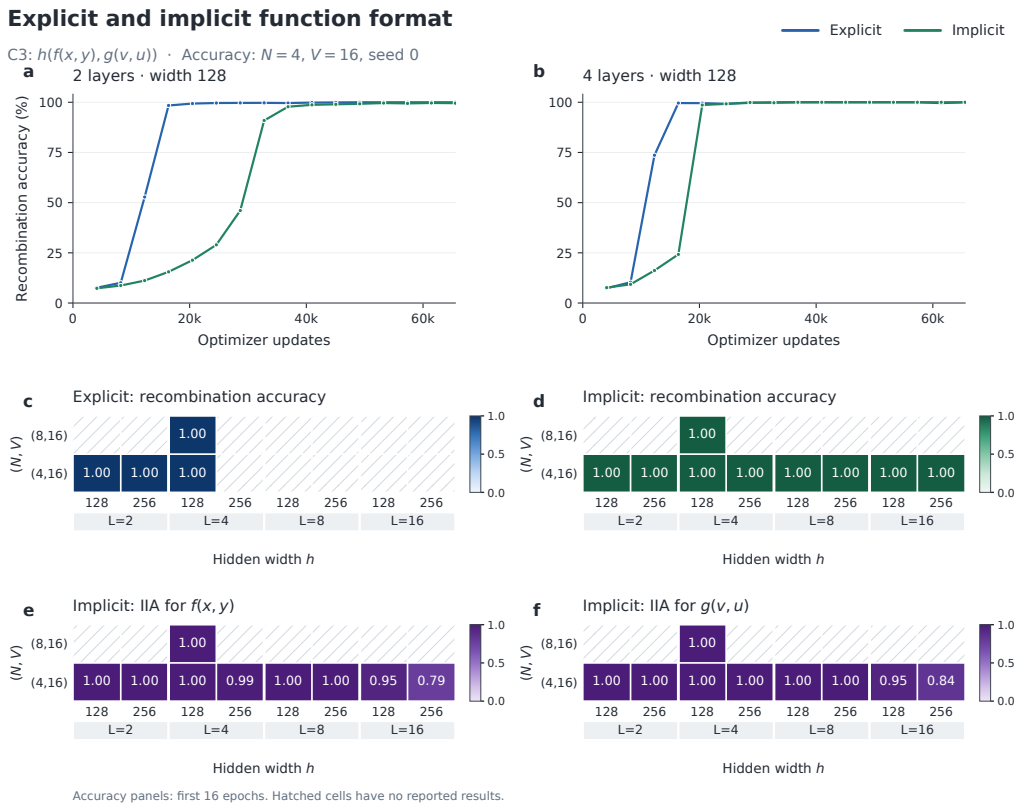


Figure 19: **Learning, generalization, and localization without explicit function labels.** (a,b) recombination accuracy during training for C3 at $(N, V) = (4, 16)$, with width $h = 128$, depth $L = 2$ or 4, and initialization seed 0. The explicit and implicit models use the same training examples. In these settings, implicit models require more training to reach high recombination accuracy. (c,d) Mean recombination accuracy for explicit and implicit models. (e,f) Mean full-rank interchange intervention accuracy for $f(x, y)$ and $g(v, u)$ in the implicit models.

C Mechanisms

This appendix gives the evidence behind Section 3.1 and Section 3.2. For each intermediate variable, we test the following on every component of the Transformer block: from which component can the value first be decoded, where do inputs with the same value form clusters, and where does replacing the component change the answer according to the replaced value. Figure 21 collects the decoding and intervention results for B and C_3 on a common set of components, and Figure 22 shows the geometry at three of them.

Model and data. We train every model from scratch on final answers only, with no supervision of intermediate values. In this analysis, we use a one-layer GPT-2 (hidden size 256, four heads) for B and two-layer OLMo2 models (hidden size 128, eight heads, gated MLP of width 1024) for other

combinators. These analyzed models reach 100% training accuracy and near-perfect recombination accuracy. For every intermediate variable we fix a set of inputs to the inner function—(f, x) pairs for B, (f, x, y) or (g, v, u) triples for C_3—and pair each of them with the same set of downstream contexts: the outer function and, for C_3, the other branch. For B we use the ten values 0–9 (640 pairs $\times$ 8 outer functions); for C_3 all 16 values (2048 triples per branch $\times$ 4 contexts). We read all activations at the position where the model can first compute the intermediate variable: x for $f(x)$, and the last argument y or u for $f(x, y)$ and $g(v, u)$.

C.1 METHODS

Decoding. We first test whether the value can be read from a component. At each component of the first block—the attention weights on the function token, all attention weights of the query, the attention output, the MLP input, the activated gate, the gated product, the MLP output and the block output—we fit a linear classifier of the intermediate value on half of the inner functions and test it on the individual contexts of the other half. The classifier assigns an activation to the value whose class mean is nearest, a linear rule with one direction per value. Because the down projection D is linear, the classifier fit at the MLP output is also a linear classifier of the gated product, and we report it there as well (Figure 21a). High accuracy shows that the value can be decoded linearly from that component, but it does not show that the model uses it.

Geometry. We average each (f, x) pair or (f, x, y) triple over its downstream contexts, plot the first two principal components of these averages colored by value (Figure 22), and report the share of their total variance that lies between values.

Causal intervention. We take a base prompt and a source prompt with different intermediate values, replace one first-block component at the argument position by its activation on the source prompt, and keep the base prompt, the parameters and all other components fixed. Success shows that the component carries the value in a form the rest of the model uses. We also apply DAS to learn a low-rank subspace in residual block output, attention output, and mlp output (Figure 20). Localization across checkpoints are shown in Figure 24.

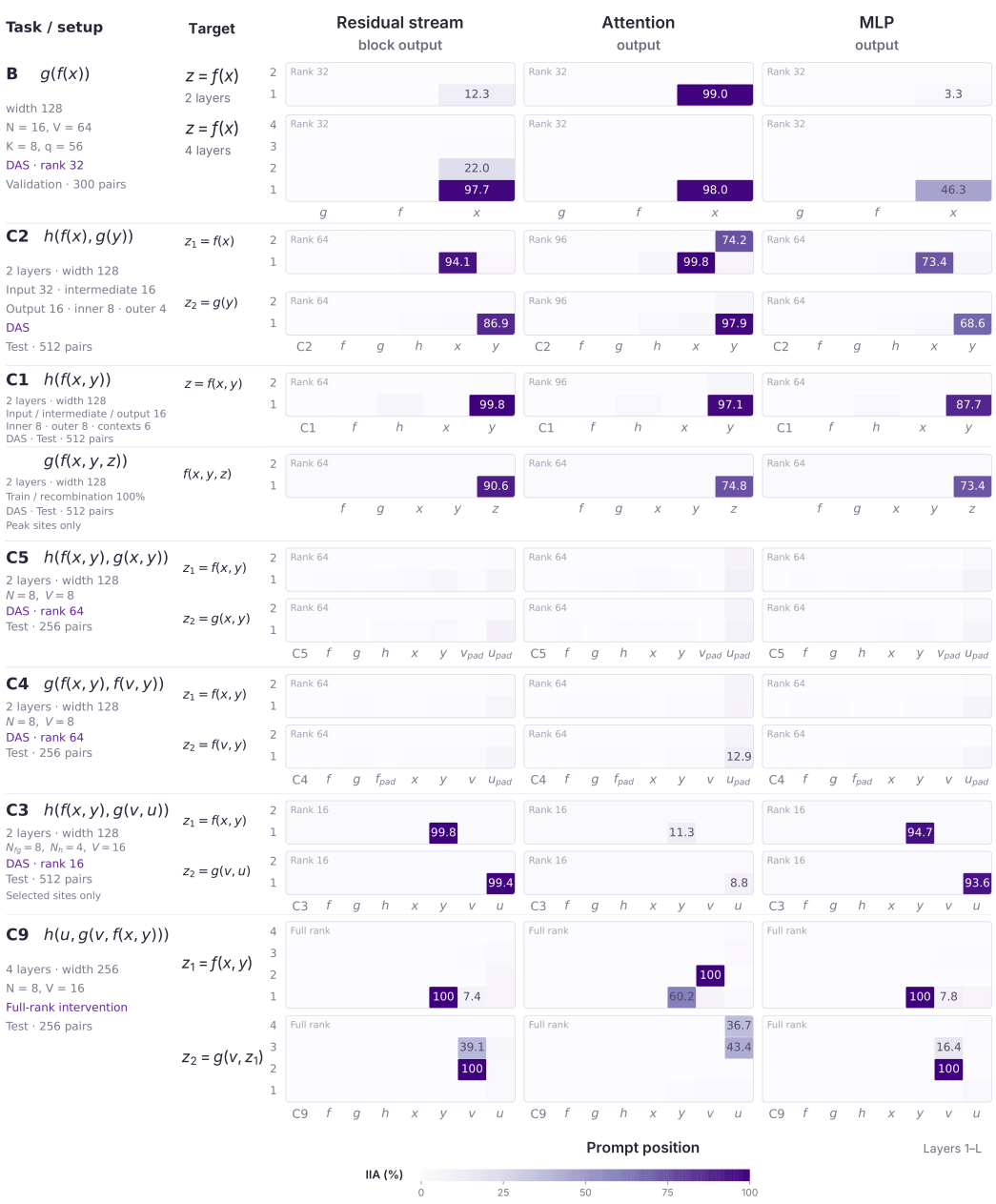


Figure 20: **Localization of intermediate variables.** Rows specify intermediate variables; columns show residual stream, attention, and MLP outputs.

C.2 Mechanism of C_3

We examine $z_L = f(x, y)$ and $z_R = g(v, u)$ in the two-layer OLMo2 model of $h(f(x, y), g(v, u))$ shown in Figures 21 and 22. The two branches follow the same mechanism, at positions y and u, respectively.

Attention and the residual deliver the function and arguments. The first-layer attention output still carries the identities of the function and its arguments: affine readouts recover the function at 97.2–97.4%, the first argument at 93.2%, and the last argument at 80.2–82.3% across the two branches. To test their routes, we replace one token's value vector by its fitting-population mean, keeping attention weights, other value vectors and the residual fixed. From 100% native answer

(a) Decoding

		attention weights on function	all attention weights	attention output	MLP input	activated gate	gated product	MLP output	block output
B $g(f(x))$	$f(x)$ at x	46.6% 1192/2560	52.4% 1342/2560	100% 2560/2560 ★	100% 2560/2560	99.3% 2543/2560	93.7% 2398/2560 (2549/2560)	93.7% 2398/2560	100.0% 2559/2560
C_3 $h(f(x,y), g(v,u))$	$f(x,y)$ at y	6.8% 278/4096	6.9% 282/4096	8.8% 362/4096	9.0% 370/4096	13.6% 556/4096	99.7% ★ 4084/4096 (2670/4096)	99.7% 4084/4096	100% 4096/4096
	$g(v,u)$ at u	6.6% 269/4096	6.3% 257/4096	9.3% 380/4096	9.4% 383/4096	13.3% 546/4096	99.7% ★ 4082/4096 (2633/4096)	99.7% 4082/4096	100% 4096/4096

(b) Interchange intervention

		attention weights on function	all attention weights	attention output	MLP input	activated gate	gated product	MLP output	block output
B $g(f(x))$	$f(x)$ at x	15.2% 39/256	34.8% 89/256	37.1%‡ 95/256	35.9%‡ 92/256	12.1% 31/256	12.9%‡ 33/256	12.9%‡ 33/256	35.9%‡ 92/256
C_3 $h(f(x,y), g(v,u))$	$f(x,y)$ at y	5.1% 13/256	10.2% 26/256	45.3%† 116/256	100% 256/256	15.2% 39/256	99.2% 254/256	99.2% 254/256	100% 256/256
	$g(v,u)$ at u	6.6% 17/256	5.9% 15/256	46.5%† 119/256	100% 256/256	9.4% 24/256	100% 256/256	100% 256/256	100% 256/256

Figure 21: **Where the intermediate value is decodable and where replacing it changes the answer.** Rows are the intermediate variables of B and C_3; columns are the components of the first block in computation order, read or replaced at the variable's position.

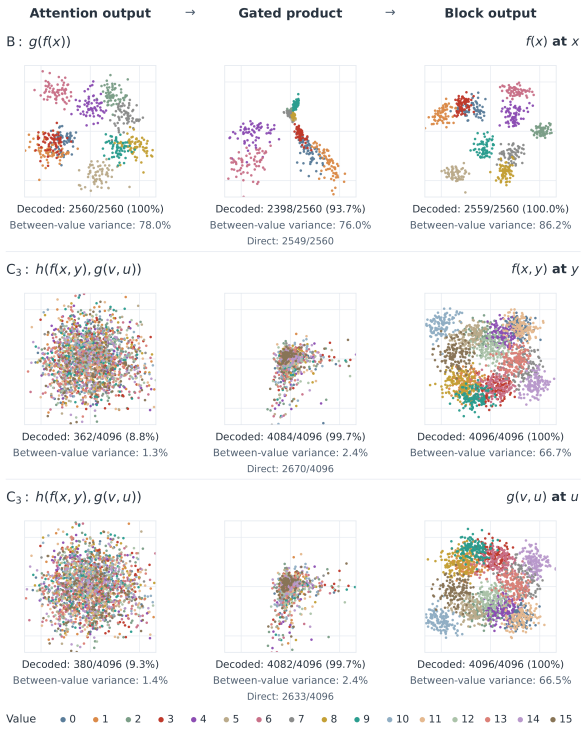


Figure 22: **Geometry and decodability of the intermediate value across components.** Each point is one (f, x) pair or (f, x, y) triple averaged over its downstream contexts and colored by its intermediate value; each panel is its own centered PCA, with symmetric axes at $1.15\times$ the 99th percentile of the coordinates, so scales differ between panels.

accuracy, averaging the function's value vector reduces accuracy to 34.8–36.1%, and averaging the first argument's reduces it to 24.9–26.2%. Averaging the last argument's value vector instead preserves 98.7–98.8%. However, replacing only its embedding contribution to the MLP input by its mean, while preserving attention, reduces accuracy to 73.0–75.4%. These results support delivery

of the function and first argument through attention values, with the last argument also contributing through the residual.

Attention weights do not give a reliable value readout. Decoding the intermediate value from the full attention-weight row gives only 6.3–6.9% accuracy, close to chance (6.25%); attention-output decoding is also low, at 8.8–9.3% (Figure 21a). Unlike the single-token case, the first-layer score between y and f does not depend on x and therefore cannot by itself encode a general $f(x, y)$. The softmax denominator can depend on x, but the tested readout does not find a strong value code in the resulting weights.

The gated product already supports an affine value readout. For MLP input r, the gated product is

$$t = \mathrm{SiLU}(W_{\mathrm{gate}} r) \odot (W_{\mathrm{up}} r),$$

where $\odot$ denotes coordinate-wise multiplication. An affine readout inherited from the MLP output through the linear down projection decodes the intermediate value at 99.7% in both branches, on held inner functions (Figure 21a). Because the down projection is linear, this readout is already available at the product. The product is thus the earliest measured stage with demonstrated near-complete affine access to both values.

Value-related geometry becomes more prominent after the down projection. At the product, between-value differences explain only 2.4% of the variance across context-averaged calls, despite the strong affine readout. At the block output, this rises to 66.7% and 66.5% for the left and right branches, and decoding reaches 100% (Figure 22). The value is therefore readable before it dominates the representation's geometry; the down projection and subsequent block operations make the value clusters more prominent.

Replacing the product transfers the intermediate value. We replace the first-layer product at y or u with that from a source prompt, keeping the base tokens, residual, other first-layer positions and parameters fixed. The target applies the base outer function to the source value of the selected branch and the base value of the other branch. Product replacement reaches this target at 99.2% for the left branch and 100% for the right (Figure 21b). Replacing only a learned rank-16 subspace of the block output, while preserving its orthogonal complement, succeeds at 100% and 99.6%, respectively. These interventions support causal reuse of both intermediate values.

The second layer combines the two values with h. We independently vary h, z_L and z_R, fitting and testing on different raw calls realizing the same combinations. At the second-layer MLP input, readouts recover h at 100%, z_L at 99.95% and z_R at 99.51%. A sum of separate contributions from these three variables explains 94.5% of attention-output variance across held combination means and 89.8% across individual held prompts. Removing the fitted h-dependent interaction from this output, while retaining raw-call deviations from combination means and the incoming residual, preserves 99.95% answer accuracy. This supports approximately additive delivery of the three ingredients. Final-answer readout then rises from 12.8% at the MLP input to 98.3% after the gate activation and 99.94% at the gated product, supporting computation of $h(z_L, z_R)$ within the second-layer MLP.

C.3 Learning dynamics

C.3.1 Gradient and representation

For a fixed set of (f, x) pairs or (f, x, y) triples averaged over their downstream contexts, we compare the loss gradient with respect to the block output with the block output itself across checkpoints (Figures 3, 4 and 23). Inputs with the same value receive the same labels under the same downstream contexts (Section 3.1.2), so their gradients can cluster before their representations do.

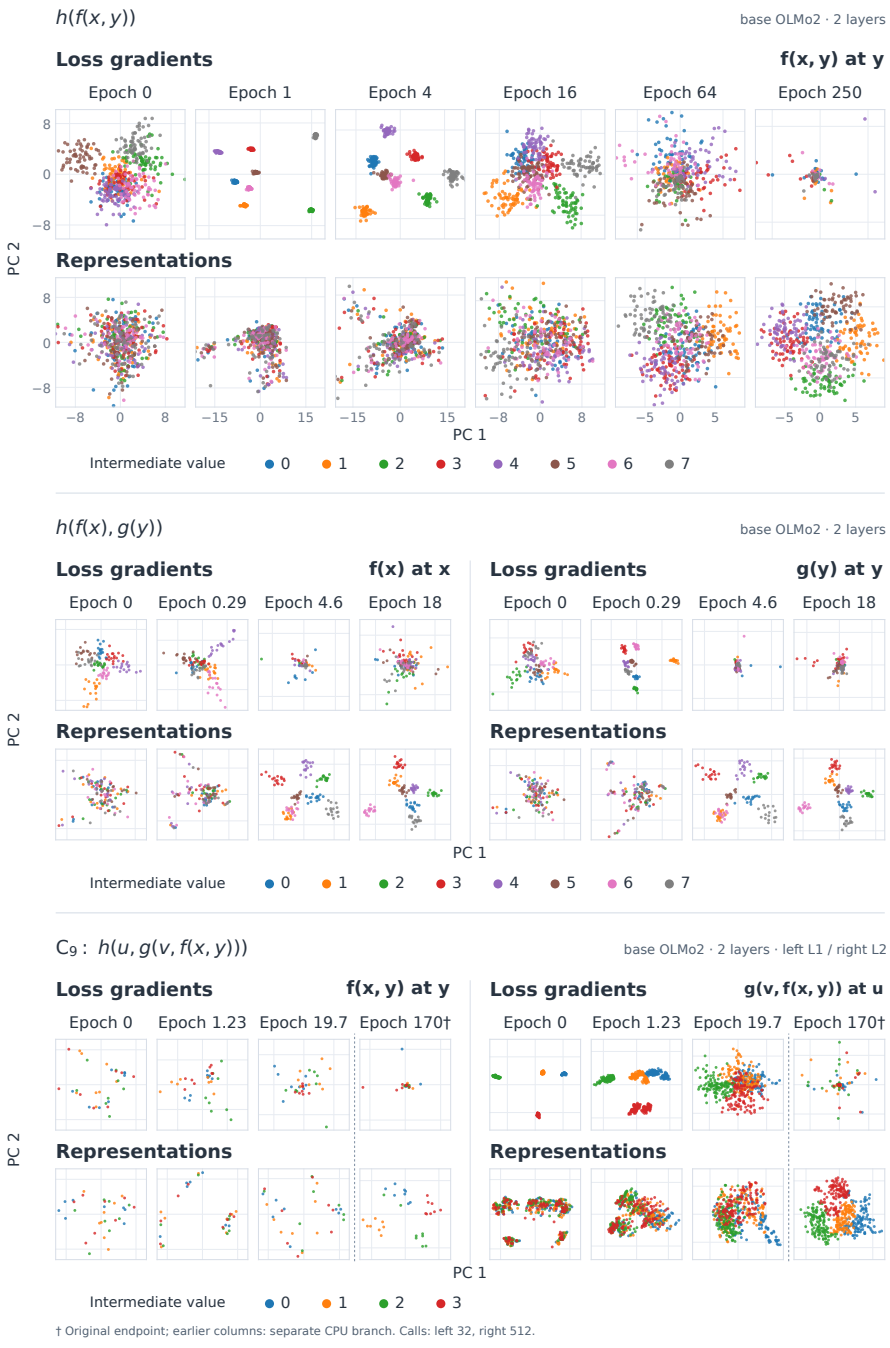


Figure 23: **Learning dynamics in more combinators.**

C.3.2 FULL LOCALIZATION RESULTS OF INTERMEDIATE VARIABLES DURING LEARNING

Figure 24 shows the learning trajectories for $C_3 : h(f(x, y), g(v, u))$, varying the number of atomic functions, model depth, and hidden width. In all ten models shown, IIA for both intermediate variables increases during early training as recombination accuracy improves. Final recombination accuracy is close to 100%, and both IIAs reach 100%. The two intermediate variables do not always reach high IIA at the same time. This trend holds for settings without explicit labels for the atomic functions (Figure 25).

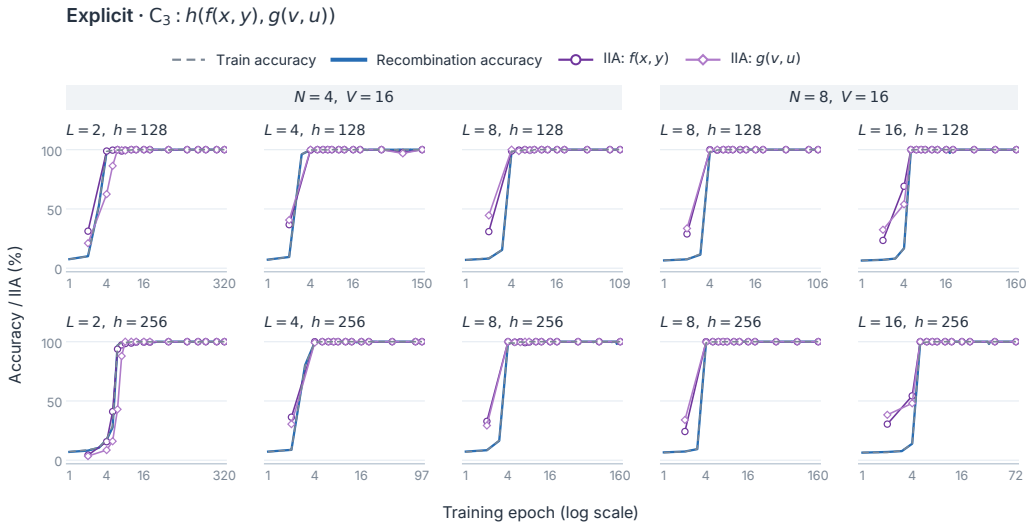


Figure 24: **Learning trajectories and localization with explicit function labels.** Train accuracy, recombination accuracy, and full-rank IIA for intermediate variables $f(x, y)$ and $g(v, u)$ in C_3. IIA is the highest score over the tested layers and positions at each checkpoint; training epochs are shown on a logarithmic scale. Columns are grouped by data configuration and model depth L; the upper and lower rows use hidden widths 128 and 256, respectively. Training uses 8,192 examples per composition at $(N, V) = (4, 16)$ and 4,096 at $(N, V) = (8, 16)$.

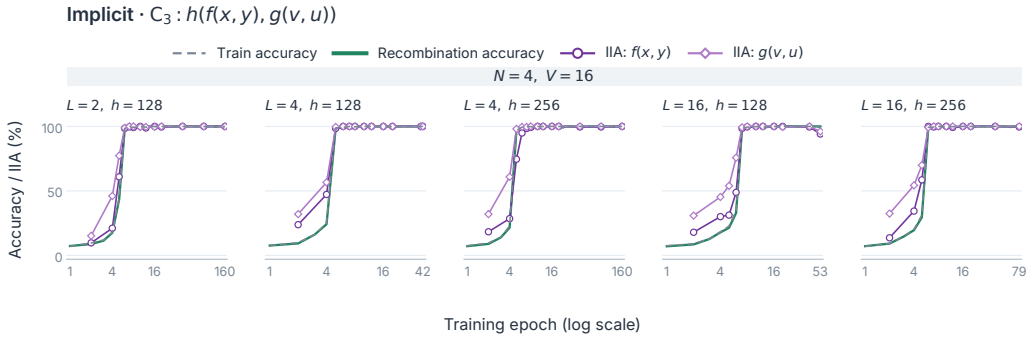


Figure 25: **Learning trajectories and localization without explicit function labels.** Train accuracy, recombination accuracy, and full-rank IIA for intermediate variables $f(x, y)$ and $g(v, u)$ in C_3. All models use $N = 4$, $V = 16$, 8,192 training examples per composition. Green shows recombination accuracy, gray shows train accuracy, and purple shows IIA.