Compositional Capabilities of Autoregressive Transformers: A Study on Synthetic, Interpretable Tasks

Rahul Ramesh [1 †] **Ekdeep Singh Lubana** [2 3 4] **Mikail Khona** [5 †] **Robert P. Dick** [2] **Hidenori Tanaka** [3 4]

Abstract

Transformers trained on huge text corpora exhibit a remarkable set of capabilities, e.g., performing basic arithmetic. Given the inherent compositional nature of language, one can expect the model to learn to compose these capabilities, potentially yielding a *combinatorial explosion* of what operations it can perform on an input. Motivated by the above, we train autoregressive Transformer models on a synthetic data-generating process that involves compositions of a set of well-defined monolithic capabilities. Through a series of extensive and systematic experiments on this data-generating process, we show that: (1) autoregressive Transformers can learn linear chains of compositions from small amounts of training data and generalize to exponentially or even combinatorially many functions; (2) generating intermediate outputs when composing functions is more effective for generalizing to new, unseen compositions than not generating any intermediate outputs (3) biases in the order of the compositions in the training data result in Transformers that fail to compose some combinations of functions; and (4) the attention layers select which capability to apply while the feed-forward layers execute the selected capability. Code is available at https://github.com/rahul13ramesh/compositional_capabilities.

1. Introduction

Large scale Transformers pretrained on huge text corpora have revolutionized machine learning in recent years (Radford et al., 2018; 2019; Brown et al., 2020; Sanh et al., 2021; Wei et al., 2021; Thoppilan et al., 2022; Touvron et al., 2023). Due to an ever-increasing interest in adopting these models in our daily lives, evaluating and predicting their capabilities has become increasingly important (Bommasani et al., 2021; Ganguli et al., 2022; Shevlane et al., 2023; Rae et al., 2021; Hoffmann et al., 2022; Tay et al., 2022; Henighan et al., 2020; Hernandez et al., 2021; Sharma & Kaplan, 2020). Motivated by this, recent works have performed extensive empirical analyses to understand the possibilities and limitations of using these models in practical tasks of interest. For example, such works show large language models (LLMs) can generate coherent text completions based on a provided context, perform code generation and debugging, use online APIs and tools in an automated manner, and even solve multimodal problems such as image captioning (Wei et al., 2022a; Bubeck et al., 2023; Austin et al., 2021; Chen et al., 2021; Lee et al., 2023; Liang et al., 2022; Qin et al., 2023; Liu et al., 2023; Suzgun et al., 2022; Srivastava et al., 2022).

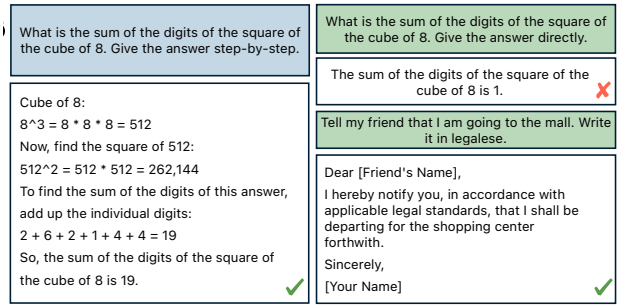


Figure 1: **Signatures of compositionality.** Language models have shown a remarkable ability to compose different capabilities (Bubeck et al., 2023) and correctly respond to prompts that require compositions of atomic arithmetic capabilities (sum, cube, square)—we argue these prompts are unlikely to be in the training data since there are a combinatorial number of combinations of capabilities (Arora & Goyal, 2023). However, the model does not always compose reliably (top-right panel). This motivates us to study the extent to which a Transformer can learn to compose its capabilities by mere pretraining on a synthetic data generating process with compositional structure.

[†]Work done during internship at NTT Research [1]Computer and Information Science, University of Pennsylvania [2]Electrical Engineering and Computer Science, University of Michigan [3]Physics& Information Labotratories, NTT Research [4]Center for Brain Science, Harvard University [5]Physics, MIT. Correspondence to: Rahul Ramesh <rahulram@seas.upenn.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

While such benchmarking of pretrained models is extremely valuable, it often focuses on evaluating rather "narrow" or "atomic" capabilities; for example, the ability to identify whether a given passage of text is biased or toxic (Gehman et al., 2020; Liang et al., 2022). However, given the compositional nature of training data (such as language), a model could learn to *compose* its atomic capabilities and perform complex tasks that it was never explicitly trained for. This can lead to an underestimation of the capabilities of the model; vice versa, if the model does not learn to compose, we can be certain that benchmarking for atomic capabilities is sufficient to characterize the model.

Motivated by the above, we analyze if a Transformer trained on a compositional data-generating process, without any special modifications to the usual training pipeline, can learn *both* relevant atomic capabilities and an ability to compose those capabilities. Bubeck et al. (2023) recently show that LLMs exhibit "sparks" of such compositionality, e.g., generating text that merges content of varying styles or evaluate mathematical expressions through the application of a sequence of functions (Fig. 1). However, due to their black-box nature, it is unclear if an LLM actually learns to compose capabilities or merely memorizes relevant samples from its training data. Moreover, while interacting with an LLM, it can be difficult to guarantee that we are utilizing a prompt that will appropriately guide the model to use the capabilities we desire, let alone compose them.

To circumvent challenges faced with LLMs pretrained on real world data and focus on our specific motivation, *"can an autoregressive Transformer trained on compositional data learn to compose its capabilities"*, we choose to limit the purview of this work to a well-defined synthetic domain. This is similar in spirit to recent works that utilize synthetic datasets generated using objects like first-order logic machines, context-free grammars, linear regressors, modular arithmetic, and even board games to establish and understand phenomenology of modern neural networks (Liu et al., 2022; Allen-Zhu & Li, 2023c;a;b; Garg et al., 2022; Li et al., 2023c; Saparov & He, 2022; Chan et al., 2022; Bhattamishra et al., 2020; Zhou et al., 2023; Nanda et al., 2023a;b; Li et al., 2023a; Lubana et al., 2023; Jones, 2021). The goal of such works, including ours, is to develop interpretable demonstrations and mechanistic hypotheses that enable a characterization of the target phenomenology in a controlled setting. Accordingly, we emphasize that we do not intend to develop novel protocols for improving Transformers' ability to compositionally generalize, but rather to demonstrate its existence and understand what drives it. Overall, we make the following contributions.

- **A minimal synthetic setup for characterizing Transformers' ability to compose.** We propose a minimal setup involving linear chains of compositions of predefined functions $\mathcal{F}$ (bijections and permutations) that operate on a string of arbitrary tokens (Section 3), which allows us to precisely study the ability of Transformers to compose functions. Motivated by instruction induction and tuning in LLMs (Honovich et al., 2022; Wei et al., 2021), we instantiate a notion of "task tokens" which specify what functions are to be applied to the input string. This helps us avoid any ambiguity in task-specification (Shah et al., 2022).
- **Transformers show explosion of capabilities.** We characterize the ability of a Transformer trained on our proposed setup to compositionally generalize, i.e., to apply a composition of specific functions chosen from $\mathcal{F}$, to an input string. We show that a Transformer, trained on very few compositions, can generalize to exponentially or even combinatorially many functions (Section 4.1)—these functions are entirely "out-of-distribution", i.e., the model never sees them in its training data and hence was not explicitly trained to learn them. Crucially, allowing the model to recursively process its intermediate outputs—i.e., stepwise inference (Kojima et al., 2022; Wei et al., 2022b)—significantly improves compositional generalization (Section 4.3 and appendix C.3).
- **Characterizing limitations and mechanisms of compositionality in a Transformer.** We formalize a notion of "distance" between the functions seen by the model during pretraining and the ones it is evaluated on, hence enabling a precise characterization of when the model struggles to compose (Section 4.2). As we show, the training data determines whether the Transformer generalizes to an exponential or combinatorial set of functions—which we call in-order and and out-of-order generalization respectively. Furthermore, linear probing (Tenney et al., 2019; Li et al., 2023a), and an analysis of the attention maps suggests the following mechanism for solving our task: the attention layer selects the task token and the fully connected layers compute the function corresponding to it (Section 4.4). We also prove the existence of Transformers that can compositionally generalize to our task and analyze why stepwise inference helps with it (Appendix C). Our mechanistic analysis and theoretical construction align extremely well.

2. Related Work

Capabilities in a Transformer. Transformers pretrained on large-scale, web-crawled datasets have been shown to exhibit a slew of interesting capabilities, such as basic arithmetic, question answering, commonsense knowledge reasoning, stylistic transformation of a piece of text, and even multimodal reasoning (Radford et al., 2018; 2019; Brown et al., 2020; Bubeck et al., 2023; Wei et al., 2022a; 2021;

Rae et al., 2021; Chowdhery et al., 2022; Austin et al., 2021; Chen et al., 2021; Bommasani et al., 2021). However, this generality can come at the cost of a model also learning capabilities that are undesirable (Bommasani et al., 2021; Tamkin et al., 2021; Chan et al., 2023), e.g., producing sensitive, biased, or toxic outputs (Weidinger et al., 2021; McGuffie & Newhouse, 2020; Garrido-Muñoz et al., 2021; Lin et al., 2021; Jiang et al., 2021; Abid et al., 2021; Parrish et al., 2021; Xu et al., 2021; Huang et al., 2019; Sheng et al., 2019; Gehman et al., 2020; Xu et al., 2020; Tamkin et al., 2021). This has motivated several works focused on understanding capabilities of a pretrained model, including (i) *predicting* capabilities of a *future* model, e.g., via fitting power laws to data/model scaling results (Rae et al., 2021; Hoffmann et al., 2022; Hernandez et al., 2021; Sharma & Kaplan, 2020; Arora & Goyal, 2023) and (ii) *eliciting* capabilities of a *given* model, e.g., via identification of appropriate prompts or via step-wise inference protocols such as chain-of-thought, to understand what tasks a model can be reliably used for (Liang et al., 2022; Suzgun et al., 2022; Lee et al., 2023). However, we argue that measuring a language model's performance on benchmarks to identify the existence of a set of capabilities is bound to be insufficient for characterizing what tasks it can perform: given the compositional nature of data these models are trained on, it is possible that they learn to *compose* capabilities, hence learning to perform several more tasks than we explicitly train them on. In fact, with a related motivation, Yu et al. (2023) design a benchmark for evaluating a model's ability to combine its skills in a recent contemporary work.

Compositionality in neural networks. The ability to compositionally reason has been touted as a cornerstone of human intelligence (Fodor & Lepore, 2002; Fodor & Pylyshyn, 1988; Fodor, 1975; Schulz et al., 2016). Accordingly, several works have studied the ability of a neural network to compositionally generalize, usually demonstrating a negative result, and correspondingly developing explicit strategies that help improve the model's ability to generalize (Liška et al., 2018; Hupkes et al., 2018; Lake & Baroni, 2018; Csordás et al., 2021b;a; 2022; Ontanón et al., 2021; Lepori et al., 2023; Lewis et al., 2022; Yun et al., 2022; Okawa et al., 2023; Hosseini et al., 2022). Our work differs from prior literature in several ways. First, we do not intend to develop protocols for improving compositional generalization in a Transformer; instead, we show that Transformers can learn to compose its capabilities and perform tasks it was never explicitly trained on, with autoregressive training on tokens from a compositional data-generating process. To this end, we define a synthetic task that allows for perfect task specification and which avoids ambiguity from prompt misspecification. While similar to the compositional table lookup task used in prior work (Liška et al., 2018; Csordás et al., 2022), our task involves a much larger set of capabilities to train and test for (3125 or 4 million, depending on the setup, compared to 128 capabilities in prior work). Second, we aim to understand the extent of compositional generalization in a Transformer trained on our proposed domain, i.e., what kind of compositions does the model fail to perform and when. We define a framework to precisely characterize these failures modes and use the popular linear probing protocol for understanding model internals to show the critical role of attention layers in enabling compositionality (Li et al., 2023a). Finally, we analyze the impact of step-wise inference protocols, wherein intermediate outputs generated by the model are recursively passed to it as inputs, and which has been used for solving several challenging benchmark tasks recently (Suzgun et al., 2022; Wei et al., 2022b).

Similar to our work, Li et al. (2023c) study step-wise inference in Transformers trained on synthetic data from a compositional data generating process. However, there are notable differences—we show that Transformers compositionally generalize to combinatorially many new functions and carefully controlling the training data allows us to highlight the benefit of step-wise inference. Furthermore, Li et al. (2023b) study compositionality with prompts used for in-context learning (Garg et al., 2022), while our synthetic setup avoids ambiguity in specifying the compositions. Many other works that study whether Transformers can compositionally generalize (Csordás et al., 2021a; Ontanón et al., 2021), focus on compositionality within a single forward pass, i.e., the model is not allowed to recursively process its inputs. We find the use of intermediate outputs significantly simplifies the problem and, given its popularity in practical scenarios (Kojima et al., 2022; Wei et al., 2022b), our results serve as a demonstration that inference protocols that allow Transformers to recursively refine their outputs can lead to a wide range of capabilities, especially ones that we never explicitly train the model for. Finally, our work studies compositions of linear chains of functions such as $F_1 \circ F_2 \circ F_3$, but compositions can be defined over more complex structures like trees (Andreas, 2019) which include functions like $F_3(F_2(\cdot), F_1(\cdot))$. Murty et al. (2022) seek to characterize such tree-structured compositional derivations implemented by a Transformer, however, they only consider a single forward pass through the Transformer. In contrast to Murty et al. (2022) our goals are different, which is to understand properties of the training data that allow us to successfully compose functions.

3. Formalizing capabilities and compositions

As noted by Hupkes et al. (2020), despite extensive work exploring compositionality in neural networks, the term is often used for several related concepts. To avoid ambiguity, we thus present a definition of a "compositional model" that

captures our intended notion and, correspondingly, describe the data-generating process used in this work to understand Transformers' ability to compose. Let $\mathcal{F}$ denote a set of predefined automorphisms, i.e., any given function F from the set defines a map between points from its input space to the same space. This is motivated by the fact that the input and output domain of a language model are generally the same. We define an **input** x as a combination of two strings $[x_f, x_d]$, where $x_f \in X_f^L$ is a sequence of L tokens that specify a series of L functions from $\mathcal{F}$, and $x_d \in X_d^K$ denotes a sequence of K tokens to which the series of L functions are applied to. We refer to x_f as **task tokens** and to x_d as **data tokens**. For example, let x_{F_i} be the identifier that denotes that function F_i is applied to the data tokens and x_{d_k} denote the k^{th} token from the vocabulary X_d. Assume $L = 2$ and $k = 1$ and define a sample $x = [x_{F_1}, x_{F_2}, x_{d_1}]$. Then, a model $M : X_f^L \times X_d^K \mapsto X_d^K$ that takes x as input, is expected to produce the output $F_2 \circ F_1\,(x_{d_1})$. We use $[L]$ to denote the ordered set $(1, 2, \ldots, L)$.

A **capability**, in our setup, is defined as the ability of a model to accurately represent a function $F \in \mathcal{F}$. We emphasize that we do not expect pretrained models in practice to perfectly implement an arbitrary function; however, this idealized definition affords us precision and allows us to use accuracy over a random set of inputs to claim a model possesses a certain capability. Based on this definition, we intend to understand the set of capabilities—or the set of functions—that a Transformer can implement by composing them. In this work, we restrict our attention to the ability to implement linear chains of compositions which we formalize as follows.

Definition 3.1 (Compositionality on linear chains.). We say a model $M(.)$ compositionally generalizes to linear chains if, for any subset of functions $F_i \in \mathcal{F}$, where $i \in [L]$, $M\left([x_{F_1}, x_{F_2}, \cdots x_{F_L}, x_d]\right) = F_L \circ \cdots \circ F_2 \circ F_1\,(x_d)$.

In practical scenarios, we would not expect the pretraining data to present a capability in all possible scenarios that it can be used in. For example, simple arithmetic tasks like multiplication are often only seen in the context of numbers with 1–3 digits in web-crawled data (Razeghi et al., 2022), which leads to an inability of the model to perform multiplication in higher order numbers. To model this in our setup, we create a spurious correlation between a subset of the functions from $\mathcal{F}$ and the position of their identifiers in the task tokens x_f. Specifically, we define $\mathcal{F}^{(l)} \subset \mathcal{F}$ as the set of functions that are allowed at the **position** l in the task tokens x_f. We let $|\mathcal{F}^{(l)}| = N$ for all locations l, i.e., $\mathcal{F}$ is partitioned into equally sized subsets and $|\mathcal{F}| = N \times L$. The notation $F_i^{(l)}$, where $i \in [N]$ and $l \in [L]$, is used to denote the i^{th} possible function at position l. Based on the above, we define two ways to compose L functions: **in-order** and **out-of-order** (see Fig. 2).

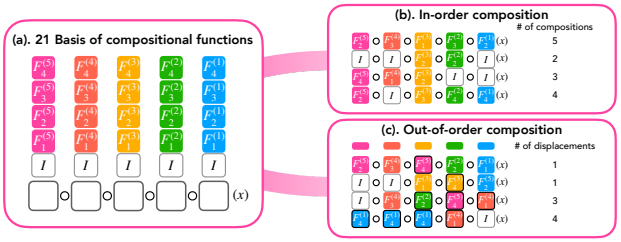


Figure 2: **Data generating process for in-order and out-of-order compositions.** (a) Each of the $L = 5$ positions is associated with $N = 4$ functions $f_i^{[l]}$, in addition to an identity function, resulting in a total of $5 \times 4 + 1 = 21$ basis functions for composition. (b) The in-order compositions select functions within the same position while (c) out-of-order compositions allow for selecting functions across positions. Each position also includes the identity function since it allows us to compute compositions of fewer than 5 functions. In the examples presented in (c), displaced functions are surrounded by a black line, and we then count the number of displaced functions.

Definition 3.2 (In-order vs. out-of-order Compositions.). Consider the composition $\widetilde{F} = F^{(l_1)} \circ F^{(l_2)} \circ \cdots \circ F^{(l_L)}\,(.)$, where $l_i \in [L]$. Denote the ordered set $(l_1, l_2, \ldots, l_L)$ as $\texttt{order}(\widetilde{F})$. If $\texttt{order}(\widetilde{F})$ equals the set $[L]$, we say $\widetilde{F}$ is an *in-order* composition; else, we say it is *out-of-order*.

To understand the number of in-order and out-of-order compositions, let us consider a composition of L functions and L sets of functions denoted by $\mathcal{F}_1, \cdots, \mathcal{F}_L$. Each set contains exactly N functions, i.e., $|\mathcal{F}_i| = N$. For an in-order composition, the function at position i (ith step of the composition) belongs to the set $\mathcal{F}_i$. There are a total of N^L in-order compositions since each position has N choices. On the other hand, for an out-of-order composition the function at position i, need not belong to $\mathcal{F}_i$, but can instead belong to any of the L sets of functions, totalling to $(N \times L)^L$ out-of-order compositions.

Consider a model M that perfectly encodes all $N \times L$ functions from the set $\mathcal{F}$. If the model can generalize to *in-order* compositions of these functions, then its set of capabilities will in fact grow to exponentially many functions (N^L). Further, the ability to compose *out-of-order* can increase this set combinatorially, i.e., proportional to $(N \times L)^L$, growing even more quickly compared to the set of in-order compositions. Such an "explosion of capabilities" would imply that it is difficult to characterize the set of all tasks that a pretrained model can perform, especially since the pretraining data used for training a model is generally unknown and hence it is hard to even characterize what "atomic" capabilities the model possesses. In our experiments, we find that while Transformers can generalize to both in-order and out-

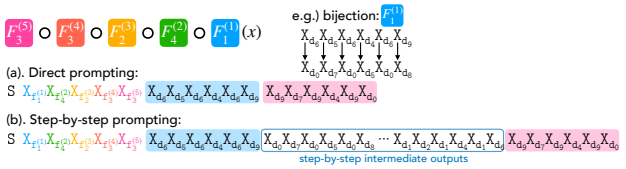


Figure 3: **Direct v.s. Step-by-step prompts.** The task (rainbow) and data (blue) tokens can be completed in two ways. They are followed by: **(a)** the intermediate outputs of the composition in the step-by-step format or **(b)** directly by the final result of compositions in the direct format.

of-order compositions, the pretraining dataset for enabling out-of-order generalization must exhibit some—albeit not huge—diversity (we quantify this further when discussing our results). To empirically characterize out-of-order compositions and discuss the failure modes thereof, we find it useful to define the following notion of **displacement** (see Fig. 2).

Definition 3.3 (**Displacement.**). Let $D(s, s')$ denote the hamming distance between two ordered sets s and s'. Then, the displacement of a composition $\widetilde{F}$ is defined as $D(\texttt{order}(\widetilde{F}), [L])$.

3.1. Experimental Setup and Data-Generating process

Having defined our notion of compositionality in a pretrained model, we now briefly discuss the experimental setup used in this work (see Appendix A for details). Specifically, our data-generating process yields inputs consisting of a sequence of 6 **data tokens**, $x_d \in X_d^6$, where each token is drawn from a vocabulary of size $|X_d| = 10$. Each of the 6 elements are drawn uniformly at random, with replacement, from X_d. We consider **two families of functions** defined over these data tokens: bijections and permutations (see Fig. 10). Specifically, the set $\mathcal{F}_b$ (which we refer to as bijections) consists of all functions that apply a bijection on each of the 6 tokens in an element-wise manner. The number of such functions is the number of bijections on a single token: there are 10! such functions when $|X_d| = 10$. The second set is $\mathcal{F}_p$, which is the set of all permutations of 6 elements ($|\mathcal{F}_p| = 6!$). The rationale for selecting these function families is that both $\mathcal{F}_b$ and $\mathcal{F}_p$ are groups with function composition as the group operator. Consequently, the composition of two functions is also a group element.

We consider two formats for representing a sample (see Fig. 3). Both formats start with task tokens x_f, that specify the sequence of functions to compose, followed by the data tokens x_d. The **direct prompt** format follows this with the final output of the function composition, while the **step-by-step prompt** format follows this with all intermediate outputs of the function composition, similar to chain-of-thought and related protocols (Kojima et al., 2022; Nye et al., 2021; Wei et al., 2022b).

We also control the set of **task tokens** seen during training. In particular, we control compositions in the training data to either only contain in-order compositions, or also include out-of-order compositions. The training data for **random** contains task tokens corresponding to a random subset of the set of all possible in-order compositions. The training data for **base** contains task tokens where at most one position in the composition is not the identity function. For example, if we consider $N = 4$ and $L = 5$ like in Fig. 2, then **base** contains compositions of functions where at least four of the five positions are identity, totalling to overall 21 functions. The set of functions **base** helps us assess whether mere learning of "atomic" capabilities is sufficient to yield compositionality in a model. (See Appendix A.2)

We generate 100K samples using the process above for a given prompt format (step-by-step or direct) and with restrictions on the task tokens (in-order, out-of-order, **base**, **random**). The model is autoregressively trained on this data using the cross-entropy loss (see Appendix A). After training, we evaluate whether the model possesses a capability corresponding to a set of composition of functions, by computing the accuracy of the model completion on 1000 different data tokens (see Appendix A.2). The accuracy of a completion is the average accuracy over the last 6 tokens.

4. Results

In this section, we systematically investigate the capabilities of an autoregressive Transformer trained on synthetic tasks with compositional structure. Broadly, we would like to understand how this structure in the data manifests in the network. We focus on addressing the following questions:

(1) Do Transformers compositionally generalize to functions not present in the training data and to what extent do they exhibit in-order and out-of-order generalization?

(2) How do properties of the training data influence in-order and out-of-order generalization?

(3) Are there differences between direct and step-by-step prompt formats?

(4) Do Transformers first learn to compose fewer functions before learning to compose many of them?

(5) What is the role of the attention and feed-forward layers?

(6) Can another popularly used architecture for autoregressive modeling, e.g., LSTMs, compositionally generalize in our setup?

We use nanoGPT (Appendix A), a Transformer with 12 layers with each Transformer block identical to the one in Vaswani et al. (2017). We use the same architecture across all our experiments in this section, but provide ablations that

vary the number of layers, attention heads, and embedding dimension in Appendix B.1.

4.1. Combinatorial explosion and Exponential growth in capabilities

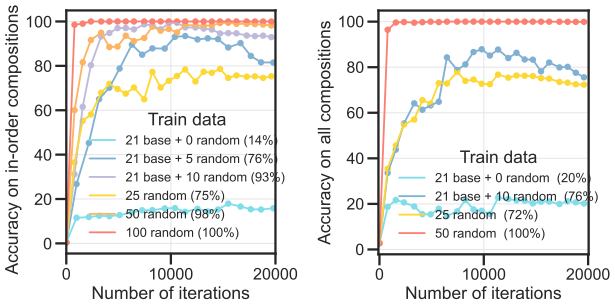
Do Transformers only generalize to functions present in the training data or do they reflect compositional structure present in data? In Fig. 4, we train on data consisting of a small subset of in-order compositions of bijections $\mathcal{F}_b$, in the step-by-step prompt format. We consider the composition of 5 functions in both Figs. 4a and 4b. Each position of the composition can be one of four choices, with the four choices at different positions being different in Fig. 4a and the same in Fig. 4b. In addition, any position can also be selected to be identity.

We find that Transformers can capture the compositional structure in data and generalize to exponential and combinatorial sets of functions in Figs. 4a and 4b, despite being trained on an *extremely small* subset of function compositions. For example, a Transformer trained on 30–100 function compositions, generalizes to 3125 unseen compositions of these functions almost perfectly.

In contrast, we note that LSTMs fail to compositionally generalize in this same setup (Appendix B.2), while Transformers with different numbers of layers and attention heads show compositional generalization (Appendix B.1). This indicates that the **inductive bias of the architecture contributes to compositional generalization and any autoregressive model is not guaranteed to succeed.** We also observe that **base**—which serves as a null model that only trains on the atomic capabilities (or functions)—does not compositionally generalize. Overall, then, we note that compositional generalization occurs with the step-by-step prompt format, provided the right architecture and training data are used.

4.2. In-order vs. Out-of-order generalization

How do biases in the training data influence a Transformer's ability to compose? Are Transformers capable of both in-order and out-of-order generalization or does it depend on the nature of training data? For the functions in Fig. 4a, the number of in-order compositions is $5^5 = 3125$ and the number of out-of-order compositions is a whopping $(21)^5 = 4084101$; essentially all of these functions are different from the ones seen in the training data. Like in Section 4.1, we only consider Transformers trained with the step-by-step prompt format on functions from the set of bijections $\mathcal{F}_b$. In Fig. 5, we consider the training data to have functions from **base**, some in-order and some out-of-order compositions. We fail to see in-order or out-of-order generalization unless the data also includes in-order or out-of-order compositions respectively. **However, a small number of in-order (10 of them) or out-of-order compositions (100 of them) in the training data results in in-order generalization or limited out-of-order generalization.** All scenarios in Fig. 5 do not fully generalize to out-of-order compositions. This indicates that out-of-order compositions may require a lot more data compared to in-order compositions.



(a) (b)

Figure 4: **Transformers trained on the step-by-step format can generalize to an exponential (a) or combinatorial (b) number of new functions.** We plot the accuracy averaged over all compositions of $L = 5$ bijections, where each position of composition has 4+1 choices, with one of them being the identity function. Each curve corresponds to training data generated by a different subset of functions and the model is trained using the step-by-step prompt format. **(a)** The choice of 5 functions are different at different positions of composition—there are 21 different functions which can be composed (in-order) in 3125 different ways. **(b)** The choice of 5 functions are identical across all 5 positions of the composition which means there are 3125 different ways to compose them; only 1365 of them are unique. Both figures are evidence that one can train on a small number of compositions of functions (around 31-100) and generalize to exponentially (a) and combinatorially (b) many functions that would be considered "out-of-distribution".

4.3. Direct vs. step-by-step compositions

Both Sections 4.1 and 4.2 discuss experiments using the step-by-step prompt format, but do these results also hold for direct prompting? Fig. 6 (left) and Fig. 15 answer this in the negative. Specifically, in Fig. 6 (left), we consider a setup identical to Fig. 4a and train on a different number of **random** functions. **Transformers fail to generalize to new in-order compositions with direct prompting when we consider compositions of bijections from $\mathcal{F}_b$.** We observe this failure even if we train on 2000 of the 3125 possible in-order compositions of functions, i.e., *even if the data has high diversity*. In contrast, in Fig. 4a, a mere 100 compositions in the step-by-step format suffices to generalize to all possible in-order compositions.

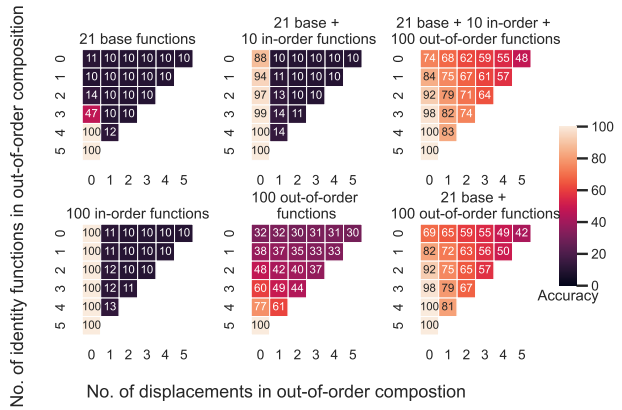


Figure 5: **The training data determines if a Transformer generalizes to an exponential (in-order generalization) or combinatorial (out-of-order generalization) number of functions.** Each sub-plot uses a different subset of functions (from $\mathcal{F}_b$) to generate the training data and we evaluate them on combinatorial set of functions generated from 20+1 functions (one of them being identity). The x-axis varies the number of displacements and the y-axis varies the number of compositions—equivalently the number of functions that are not identity. We make the following observations: (1) A Transformer trained on just 31 functions (top-middle) generalize to nearly exponentially many or 3125 compositions of functions. (2) All the above configurations do not generalize perfectly to the entire combinatorial set. They however partially generalize to nearly 4 million compositions of functions. The generalization is worse if we increase the number of compositions or displacements (see Fig. 2 for pictorial description of displacements).

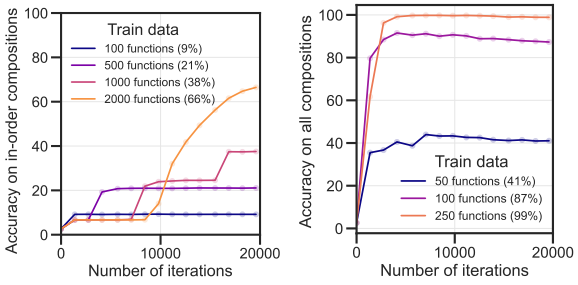


Figure 6: **Compositional generalization is less frequently seen in the direct prompt format. (Left.)** We train a Transformer using the direct prompt format on 20+1 bijections with 5 compositions with 4 choices at each position. The model fails to generalize to all 3125 compositions even if it trained on 2000 such functions. **(Right.)** We train a Transformer using the direct prompt format on a composition of two functions, with one function being one of 25 bijections and the other function being one of 25 permutations (totalling to 625 compositions). The model is able to compose previously unseen combinations of functions when trained on 250 of these functions in this scenario.

On the other hand, we see in-order generalization if a Transformer is trained on a composition of a permutation function from $\mathcal{F}_p$ and a bijection function from $\mathcal{F}_b$. In Fig. 6 (right), we train on compositions of two functions, where one position is one of 25 bijections, and the other is one of 25 permutations. We vary the number of compositions seen in the training data and find that 250 compositions in the training data are enough for the model to generalize to all 625 possible compositions of the two functions. We note that bijections and permutations operate on orthogonal features of the input: bijections operate on the value of the token while permutations operate on the position of the token. We speculate that this is important for compositional generalization in the direct prompt format.

Why is compositional generalization harder for direct prompts? (Appendix C.3) The ability to run multiple forward passes through the model allows us tackle a richer class of problems (Merrill & Sabharwal, 2023). The step-by-step and direct prompt formats differ because the former allows L forward passes through the model, while the latter only allows one forward pass. As a result, we expect for the direct prompt format to enable compositional generalization, it must compute the L steps of the composition in the intermediate layers of the model within a single forward pass itself. For example, consider a model that computes the functions F and G, and is able to compositionally generalize to function $G \circ F$. Since $G \circ F$ is computed using a single forward pass, G must occur in a layer after F (see also Fig. 11b). However, this model may not generalize to $F \circ G$, since that will require F to occur after G in it model's layers. Hence, to compositionally generalize to both combinations of F and G, a model may have to learn copies of F and G at multiple layers. This will likely require training data with large amounts of data diversity so that most combinations of functions are seen by the model during training itself.

We further formalize the intuition above in Appendix C. Specifically, in Appendix C.3, we argue that a model trained with the direct prompt format requires more compositions in the training data, by a factor of $\mathcal{O}(L)$, compared to a model trained with the step-by-step format. In Theorem C.2, we prove that there exists an L-layer Transformer that can compositionally generalize with direct prompting. However, empirically, we find that even with the additional training data, the direct prompt format fails to generalize in Fig. 6 (left). This is because the existence of a solution need not guarantee that a Transformer trained with gradient descent converges to that particular minima. The weights can instead converge to a minima that only memorizes compositions

present in the training data.

4.4. Towards a mechanistic understanding

In this section, we try to uncover the underlying mechanism for compositional generalization exhibited by Transformers in our setup—particularly for compositions of bijections in the step-by-step prompt format. Prior work on mechanistic interpretability often studies smaller neural networks to extract insights for larger networks (Nelson et al., 2021; Wang et al., 2022; Chughtai et al., 2023). The rationale relates to the universality hypothesis (Li et al., 2015; Olah et al., 2020), which states that networks of different scales are likely to learn similar functions when trained on the same data. In line with this direction, we attempt to understand a 1-layer Transformer[1] trained on our data generating process.

To develop a hypothesis for our mechanistic evaluation, we first show in Appendix C.1 the existence of 1-layer Transformers that can compositionally generalize to a simplified version of our task via the step-by-step prompt format. In particular, our construction uses the attention layer to copy the relevant task token—similar to an induction head (Olsson et al., 2022)—and the feed-forward layer to compute a single step of the function composition. The model is run L times serially, where each run computes one step of the function composition. The attention layer uses a position encoding as the key and query to determine which tokens to attend to and propagates the task token as the value.

We next evaluate if the theoretical construction, even though a simplification, lines up with empirical evaluations on the actual task. Specifically, we first use linear probing to understand which layers contribute to improvements in the accuracy and then visualize the attention maps to understand which tokens the model attends to.

Linear probe accuracy. In Fig. 7 (left), we use a linear probe to analyze the importance of attention layers and MLP layers in a 12-layer Transformer. Following Geva et al. (2022), we fix the parameters of probe to the last linear layer, i.e., the unembedding layer of the trained model. We use a Transformer trained on 100 **random** in-order compositions of 5 functions identical to the model in Fig. 4a. In Fig. 14 we show similar linear probe experiments on Transformers of different sizes and consistently find a sharp increase in accuracy right after an MLP layer, i.e., the accuracy rarely increases after an attention layer.

Visualizing attention maps. Analyzing the attention maps of a 12-layer Transformer for a discernible pattern can be difficult. We hence analyze the attention maps of a 1-layer Transformer trained for step-by-step prompts, which also exhibits in-order generalization. In Fig. 7 (right), we plot the attention map for a predefined composition of functions from the set $\mathcal{F}_b$. Keeping the task tokens to be fixed corresponding to the predefined composition, we sample 1000 data tokens and compute the attention map for the 1-layer model. The average of these maps is reported in the figure. We see that all data tokens attend to: (i) the task token that specifies the current function to be computed and (ii) the data token that the function is to be applied to.

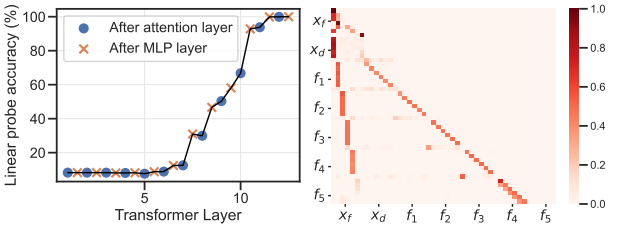


Figure 7: **(Left.) Attention layer picks a function to apply given the current input, and MLP applies the selected function for Transformers trained on compositions of bijections in the step-by-step prompt format. We see a sharp increases in accuracy after MLP layers in the last few layers of the Transformer.** We compute the linear probe accuracy—averaged over in-order compositions of functions—after the MLP and attention layers at every layer of the 12-layer Transformer. We observe a similar trend in Transformers of different depths (fig. 14). **(Right.) Attention is largest at the relevant data and task token.** We plot the causal attention mask of a 1-layer Transformer trained using the step-by-step format on compositions of 5 in-order bijections (setup of Fig. 4). Keeping the prompt fixed to a specific composition of functions, we plot the attention map averaged over 1000 samples. We observe that the current data token attends to the a specific task relevant to compute the next step of the composition.

The results above remarkably line up with our theoretical construction. For example, the attention maps in Fig. 7 always attend to the relevant task tokens and data token when computing the next step of the composition. The task and data tokens are all embedded in orthogonal spaces, similar to our construction, with the exception of 5 tokens which all correspond to the the identity function (see Appendix B.7). In parallel, the linear probe accuracy for a 1-layer Transformer in Fig. 14 shows no increase in accuracy after the attention layer (similar to results in Fig. 7), but a sharp increase in accuracy occurs after the MLP layers, indicating that the function is entirely computed in the MLP layers.

[1] In fact, we use a deeper model in most experiments in the main paper to elicit maximal performance when using the direct format; the step-by-step format, as we argue in Appendix C, can generalize compositionally with fewer layers (one, for in-order generalization).

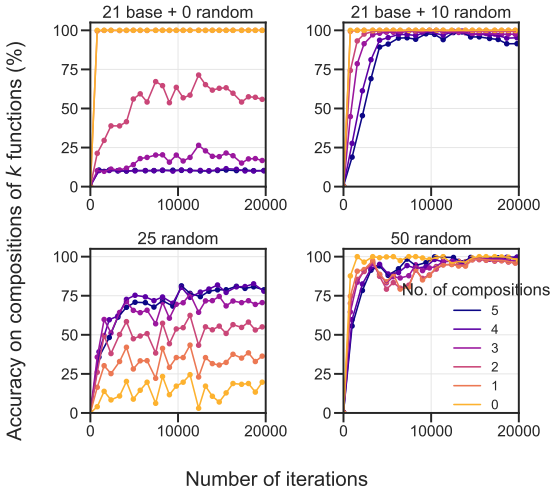


Figure 8: **Transformers trained on baseand some random functions generalizes first to a composition of more functions before it generalizes to a composition of few of them.** Each line is the average accuracy over all composition of k functions and each subplot is a Transformer trained on a different subset of functions. The **base** is trained on the individual functions and these Transformers learn to compose a smaller set of functions (more functions in composition are identity) before learning to compose many of them. The opposite is true when the model is trained on a random subset of 25 compositions of functions.

4.5. Training dynamics

We would like to study how compositional capabilities evolve over the course of training (Okawa et al., 2023) and understand if Transformers learn functions F_1 and F_2 before learning compositions like $F_1 \circ F_2$. In Fig. 8, we track the accuracy over the course of training to understand if compositions of fewer functions are learned before compositions of many functions. The setup for this figure is identical to Fig. 4a with the accuracy faceted by the number of function compositions. We find that the order in which functions are learned depends entirely on the training data. If the training data consists of **base** and very few in-order compositions, then a Transformer generalizes to fewer compositions (more identities) first before generalizing to compositions of multiple functions. On the other hand, if the model is trained on 25 **random** in-order compositions, then it is better at generalizing to more complex compositions of these functions; this trend is lost when we train on 50 **random** in-order compositions.

5. Conclusion

Given several recent works focused on prediction or elicitation of capabilities in pretrained models, we ask whether the very motivation guiding these works is tractable: can we possibly characterize all capabilities of a model, specifically a Transformer, pretrained on a compositional data domain? To address this question, we proposed a synthetic, but well-defined, data domain and formalized the notion of a capability as representing a function defined over the domain. Breaking compositional generalization into two relevant scenarios (in-order vs. out-of-order), we showed that the compositional structure of the data forces a model to learn to compose at relatively minimal data diversity, which indicatively address our primary question: an appropriate prompt could make the model compose its capabilities, yielding an "explosion of capabilities". This can arguably make tractable analysis of capabilities in a pretrained model relatively difficult.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

RR thanks Kento Nishi, Gautam Reddy and Eric Bigelow for their discussions at the early stages of this project. RR thanks AWS AI, for their gift to Penn Engineering's ASSET Center for Trustworthy AI. ESL was partially supported by the National Science Foundation (IIS-2008151).

Author Contributions

ESL and RR conceived the initial project direction and defined the problem setup with with inputs from HT and MK. The experiments were led by RR with inputs from ESL, HT and MK. The writing of the introduction and related work was led by ESL with help from HT and RR. RR, ESL and HT extensively collaborated on the methods section. The results and appendix were led by RR. The expository figures were created by HT and RR. HT and RPD acted as advisors in the work.

References

Abid, A., Farooqi, M., and Zou, J. Persistent anti-muslim bias in large language models. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 298–306, 2021.

Ahn, K., Cheng, X., Daneshmand, H., and Sra, S. Transformers learn to implement preconditioned gradient descent for in-context learning. *arXiv preprint arXiv:2306.00297*, 2023.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023a.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023b.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 1, context-free grammar. *arXiv preprint arXiv:2305.13673*, 2023c.

Andreas, J. Measuring compositionality in representation learning. *arXiv preprint arXiv:1902.07181*, 2019.

Arora, S. and Goyal, A. A theory for emergence of complex skills in language models. *arXiv preprint arXiv:2307.15936*, 2023.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bhattamishra, S., Ahuja, K., and Goyal, N. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint. arXiv:2108.07258*, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Chan, A., Salganik, R., Markelius, A., Pang, C., Rajkumar, N., Krasheninnikov, D., Langosco, L., He, Z., Duan, Y., Carroll, M., et al. Harms from increasingly agentic algorithmic systems. *arXiv preprint arXiv:2302.10329*, 2023.

Chan, S., Santoro, A., Lampinen, A., Wang, J., Singh, A., Richemond, P., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Chughtai, B., Chan, L., and Nanda, N. A toy model of universality: Reverse engineering how networks learn group operations, may 2023. *URL http://arxiv. org/abs/2302*, 3025, 2023.

Csordás, R., Irie, K., and Schmidhuber, J. The devil is in the detail: Simple tricks improve systematic generalization of transformers. *arXiv preprint arXiv:2108.12284*, 2021a.

Csordás, R., Irie, K., and Schmidhuber, J. The neural data router: Adaptive control flow in transformers improves systematic generalization. *arXiv preprint arXiv:2110.07732*, 2021b.

Csordás, R., Irie, K., and Schmidhuber, J. Ctl++: Evaluating generalization on never-seen compositional patterns of known functions, and compatibility of neural representations. *arXiv preprint arXiv:2210.06350*, 2022.

Fodor, J. A. *The language of thought*, volume 5. Harvard university press, 1975.

Fodor, J. A. and Lepore, E. *The compositionality papers*. Oxford University Press, 2002.

Fodor, J. A. and Pylyshyn, Z. W. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2): 3–71, 1988.

Ganguli, D., Hernandez, D., Lovitt, L., Askell, A., Bai, Y., Chen, A., Conerly, T., Dassarma, N., Drain, D., Elhage, N., et al. Predictability and surprise in large generative models. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1747–1764, 2022.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Garrido-Muñoz, I., Montejo-Ráez, A., Martínez-Santiago, F., and Ureña-López, L. A. A survey on bias in deep nlp. *Applied Sciences*, 11(7):3184, 2021.

Gehman, S., Gururangan, S., Sap, M., Choi, Y., and Smith, N. A. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

Geva, M., Caciularu, A., Dar, G., Roit, P., Sadde, S., Shlain, M., Tamir, B., and Goldberg, Y. Lm-debugger: An interactive tool for inspection and intervention in transformer-based language models. *arXiv preprint arXiv:2204.12130*, 2022.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Hernandez, D., Kaplan, J., Henighan, T., and McCandlish, S. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Honovich, O., Shaham, U., Bowman, S. R., and Levy, O. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*, 2022.

Hosseini, A., Vani, A., Bahdanau, D., Sordoni, A., and Courville, A. On the compositional generalization gap of in-context learning. *arXiv preprint arXiv:2211.08473*, 2022.

Huang, P.-S., Zhang, H., Jiang, R., Stanforth, R., Welbl, J., Rae, J., Maini, V., Yogatama, D., and Kohli, P. Reducing sentiment bias in language models via counterfactual evaluation. *arXiv preprint arXiv:1911.03064*, 2019.

Hupkes, D., Singh, A., Korrel, K., Kruszewski, G., and Bruni, E. Learning compositionally through attentive guidance. *arXiv preprint arXiv:1805.09657*, 2018.

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67: 757–795, 2020.

Jiang, L., Hwang, J. D., Bhagavatula, C., Le Bras, R., Liang, J., Dodge, J., Sakaguchi, K., Forbes, M., Borchardt, J., Gabriel, S., et al. Can machines learn morality? the delphi experiment. *arXiv e-prints*, pp. arXiv–2110, 2021.

Jones, A. L. Scaling scaling laws with board games. *arXiv preprint arXiv:2104.03113*, 2021.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

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

Lee, T., Yasunaga, M., Meng, C., Mai, Y., Park, J. S., Gupta, A., Zhang, Y., Narayanan, D., Teufel, H. B., Bellagente, M., et al. Holistic evaluation of text-to-image models. *arXiv preprint arXiv:2311.04287*, 2023.

Lepori, M. A., Serre, T., and Pavlick, E. Break it down: Evidence for structural compositionality in neural networks. *arXiv preprint arXiv:2301.10884*, 2023.

Lewis, M., Yu, Q., Merullo, J., and Pavlick, E. Does clip bind concepts? probing compositionality in large image models. *arXiv preprint arXiv:2212.10537*, 2022.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task, 2023a. Comment: ICLR 2023 oral (notable-top-5%): https://openreview.net/forum?id=DeG07_TcZvT ; code: https://github.com/likenneth/othello_world.

Li, Y., Yosinski, J., Clune, J., Lipson, H., and Hopcroft, J. Convergent learning: Do different neural networks learn the same representations? *arXiv preprint arXiv:1511.07543*, 2015.

Li, Y., Ildiz, M. E., Papailiopoulos, D., and Oymak, S. Transformers as algorithms: Generalization and implicit model selection in in-context learning. *arXiv preprint arXiv:2301.07067*, 2023b.

Li, Y., Sreenivasan, K., Giannou, A., Papailiopoulos, D., and Oymak, S. Dissecting chain-of-thought: A study on compositional in-context learning of mlps. *arXiv preprint arXiv:2305.18869*, 2023c.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Lin, S., Hilton, J., and Evans, O. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Liška, A., Kruszewski, G., and Baroni, M. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*, 2018.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.

Lubana, E. S., Bigelow, E. J., Dick, R. P., Krueger, D., and Tanaka, H. Mechanistic mode connectivity. In *International Conference on Machine Learning*, pp. 22965–23004. PMLR, 2023.

McGuffie, K. and Newhouse, A. The radicalization risks of gpt-3 and advanced neural language models. *arXiv preprint arXiv:2009.06807*, 2020.

Merrill, W. and Sabharwal, A. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Murty, S., Sharma, P., Andreas, J., and Manning, C. D. Characterizing intrinsic compositionality in transformers with tree projections. *arXiv preprint arXiv:2211.01288*, 2022.

Myers, A. N. and Wilf, H. S. Some new aspects of the coupon collector's problem. *SIAM review*, 48(3):549–565, 2006.

Nanda, N., Chan, L., Liberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023a.

Nanda, N., Lee, A., and Wattenberg, M. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*, 2023b.

Nelson, E., Neel, N., Catherine, O., Tom, H., Nicholas, J., Ben, M., Amanda, A., Yuntao, B., Anna, C., Tom, C., et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Okawa, M., Lubana, E. S., Dick, R. P., and Tanaka, H. Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task. *arXiv preprint arXiv:2310.09336*, 2023.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Ontanón, S., Ainslie, J., Cvicek, V., and Fisher, Z. Making transformers solve compositional tasks. *arXiv preprint arXiv:2108.04378*, 2021.

Parrish, A., Chen, A., Nangia, N., Padmakumar, V., Phang, J., Thompson, J., Htut, P. M., and Bowman, S. R. Bbq: A hand-built bias benchmark for question answering. *arXiv preprint arXiv:2110.08193*, 2021.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Razeghi, Y., Logan IV, R. L., Gardner, M., and Singh, S. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

Saparov, A. and He, H. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

Schulz, E., Tenenbaum, J., Duvenaud, D. K., Speekenbrink, M., and Gershman, S. J. Probing the compositionality of intuitive functions. *Advances in neural information processing systems*, 29, 2016.

Shah, R., Varma, V., Kumar, R., Phuong, M., Krakovna, V., Uesato, J., and Kenton, Z. Goal misgeneralization: Why correct specifications aren't enough for correct goals. *arXiv preprint arXiv:2210.01790*, 2022.

Sharma, U. and Kaplan, J. A neural scaling law from the dimension of the data manifold. *arXiv preprint arXiv:2004.10802*, 2020.

Sheng, E., Chang, K.-W., Natarajan, P., and Peng, N. The woman worked as a babysitter: On biases in language generation. *arXiv preprint arXiv:1909.01326*, 2019.

Shevlane, T., Farquhar, S., Garfinkel, B., Phuong, M., Whittlestone, J., Leung, J., Kokotajlo, D., Marchal, N., Anderljung, M., Kolt, N., et al. Model evaluation for extreme risks. *arXiv preprint arXiv:2305.15324*, 2023.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Tamkin, A., Brundage, M., Clark, J., and Ganguli, D. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv preprint arXiv:2102.02503*, 2021.

Tay, Y., Wei, J., Chung, H. W., Tran, V. Q., So, D. R., Shakeri, S., Garcia, X., Zheng, H. S., Rao, J., Chowdhery, A., et al. Transcending scaling laws with 0.1% extra compute. *arXiv preprint arXiv:2210.11399*, 2022.

Tenney, I., Das, D., and Pavlick, E. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

Weiss, G., Goldberg, Y., and Yahav, E. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Xu, A., Pathak, E., Wallace, E., Gururangan, S., Sap, M., and Klein, D. Detoxifying language models risks marginalizing minority voices. *arXiv preprint arXiv:2104.06390*, 2021.

Xu, J., Ju, D., Li, M., Boureau, Y.-L., Weston, J., and Dinan, E. Recipes for safety in open-domain chatbots. *arXiv preprint arXiv:2010.07079*, 2020.

Xu, W. and Tang, A. K. A generalized coupon collector problem. *Journal of Applied Probability*, 48(4):1081–1094, 2011. doi: 10.1239/jap/1324046020.

Yu, D., Kaur, S., Gupta, A., Brown-Cohen, J., Goyal, A., and Arora, S. Skill-mix: A flexible and expandable family of evaluations for ai models. *arXiv preprint arXiv:2310.17567*, 2023.

Yun, T., Bhalla, U., Pavlick, E., and Sun, C. Do vision-language pretrained models learn composable primitive concepts? *arXiv preprint arXiv:2203.17271*, 2022.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

A. Experimental Details

A.1. Training methodology

Transformer architecture We use nanoGPT[2] with 12 layers, 12 attention heads and an embedding dimension of size 120. Each transformer block contains a causal attention layer, layer-norms, residual connections and an MLP (see Fig. 9). The MLP contains two fully-connected layers sandwiched by a GELU layer (Hendrycks & Gimpel, 2016) The first fully-connected layers has a hidden layer with size 4 times the embedding dimension (480) and the second hidden layer has a size equal to the embedding dimension (120).

The input tokens are converted to one-hot vectors before being passed through to the model. We do not use dropout or biases in the LayerNorm layers. We use weight-tying (Press & Wolf, 2016), i.e., the input and the output embedding layers share weights. Finally, we make use of mixed-precision (bf16 in torch) to speed-up training.

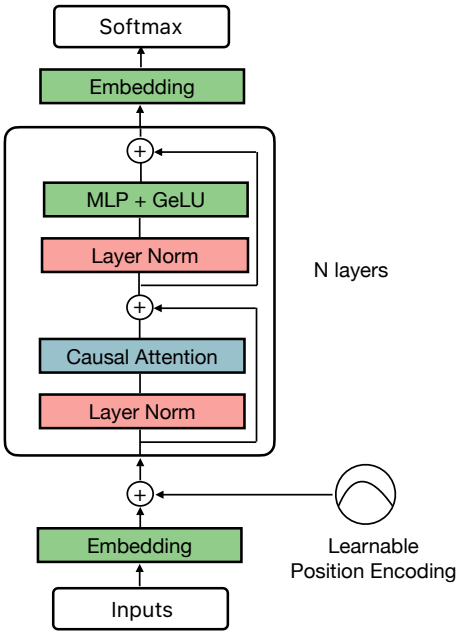


Figure 9: We use nanoGPT as the Transformer architecture in all our experiments. The core Transformer block is a LayerNorm, a causal attention block, followed by another layer-norm and a 2-layer multi-layer perceptron (MLP). The Transformer block has two residual connections.

Loss and Optimizer Models are trained using an autoregressive objective to predict the next token using the cross-entropy loss. Specifically, assume a sequence of tokens of t tokens denoted by $x_{1:t}$. Let $p_w(y \mid x_{1:t})$ denote the probability distribution over the next token as predicted by a model with weights w. For a sequence $x_{1:T}$ of length T, the autoregressive objective is

$$L(w) = -\sum_{t=1}^{T-1} \log p_w \left(y = x_{t+1} \mid x_{1:t}\right).$$

Training is performed for 100 epochs with a cosine-annealed scheduled with warmup. We use an initial learning rate of 3e-4 annealed eventually to 6e-5. We use AdamW as the optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.95$) with a weight decay of 10^{-3} and a batch-size of 512. We also make use of gradient clipping with a magnitude of 1.

A.2. Data generating process

Data and task tokens. Both data and task tokens are converted to one-hot vectors before being fed to the Transformer. The set of data tokens is denoted by X_d and the size of the vocabulary, $|X_d|$, is 10 in all our experiments. The data tokens in the input $x_d \in X_d^6$ is a sequence of 6 tokens and is the input to the function composition. The 6 tokens are sampled uniformly at random from X_d with replacement.

There are two sets of functions considered in this work. The set of functions $\mathcal{F}_b$ (which we refer to as bijections) applies a lookup table in an element-wise fashion to each of the 6 tokens in x_d. The set of functions in $\mathcal{F}_p$ permute the 6 tokens in x_d. The family of functions in $\mathcal{F}_b$ and $\mathcal{F}_p$ are described in Fig. 10. Each function from $\mathcal{F}_p$ and $\mathcal{X}_b$ has its own task token in X_F.

The input starts with a sequence of L task tokens $x_f \in X_F^L$. The number of compositions is generally $L = 5$, but in a few experiments like Figs. 15, 6 (Right), $L = 2$.

Sampling task tokens The task tokens can be sampled such that they satisfy certain properties. For example, let us consider the composition of two functions—one from the set $\mathcal{F}_1 \subset \mathcal{F}_p$ and another from $\mathcal{F}_2 \subset \mathcal{F}_b$ (which is the setting in Fig. 6 (Right)). We can restrict the training data to compositions from the set $\mathcal{F}_2 \circ \mathcal{F}_1$ which are in-order compositions

[2]https://github.com/karpathy/nanoGPT

(see Fig. 2). Alternatively, we can also choose to include out-of-order compositions, which include compositions from $\mathcal{F}_1 \circ \mathcal{F}_1, \mathcal{F}_2 \circ \mathcal{F}_2$ and $\mathcal{F}_1 \circ \mathcal{F}_2$. In Fig. 6 (Right), we restrict our training and evaluation to in-order compositions of functions and we observe that training on a subset of the elements from $\mathcal{F}_2 \circ \mathcal{F}_1$ suffices to compositionally generalize all functions in the set.

Two other commonly used subsets of functions are **base** and **random**. Consider $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_5 \subset \mathcal{F}_b$. The set **random** considers k functions from the set $\mathcal{F}_5 \circ \mathcal{F}_4 \circ \cdots \circ \mathcal{F}_1$ which are drawn uniformly at random.

base is used to test if the compositionality is seen when the Transformer is trained on the individual functions from $\mathcal{F}_i$ for all $i \in [5]$. In the training data, all compositions have 4 of the 5 functions to be the identity function I, i.e it considers compositions of the form $I \circ I \circ \mathcal{F}_3 \circ I \circ I$ or $I \circ \mathcal{F}_4 \circ \cdots \circ I$. There are a total of $1 + \sum_{i=1}^{5} \mathcal{F}_i$ such functions; the 1 is when all 5 functions in the composition are identity. The model is never trained on the composition of two or more functions, and at least compositions of 3 functions are necessary to generalize to all in-order compositions Fig. 18.

Generating a sequence of tokens A sequence starts with a sequence of two task tokens $x_f = [x_{F_1}, x_{F_2}]$ followed by a sequence of data tokens x_d. The sequence can either be presented in: (i) The step-by-step format, where the intermediate outputs are also included in the sequence; e.g., the sequence in the step-by-step format would look like $[x_{F_1}, x_{F_2}, x_d, F_1(x_d), F_2(F_1(x_d))]$ (see Fig. 11a) or (ii) The direct format, which does not include the intermediate outputs of the composition in the sequence and an example of such a sequence is $[x_{F_1}, x_{F_2}, x_d, F_2(F_1(x_g))]$ (see Fig. 11b).

The step-by-step and direct formats are also discussed in Fig. 3. The training data consists of 100,000 sequences for all experiments in one of the two formats.

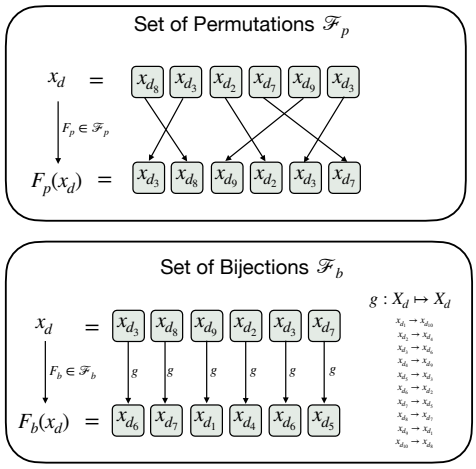


Figure 10: A permutation from $\mathcal{F}_p$ permutes the 6 tokens in the input x_d. A bijection from $\mathcal{F}_b$ applies a lookup table to each of the 6 tokens individually.

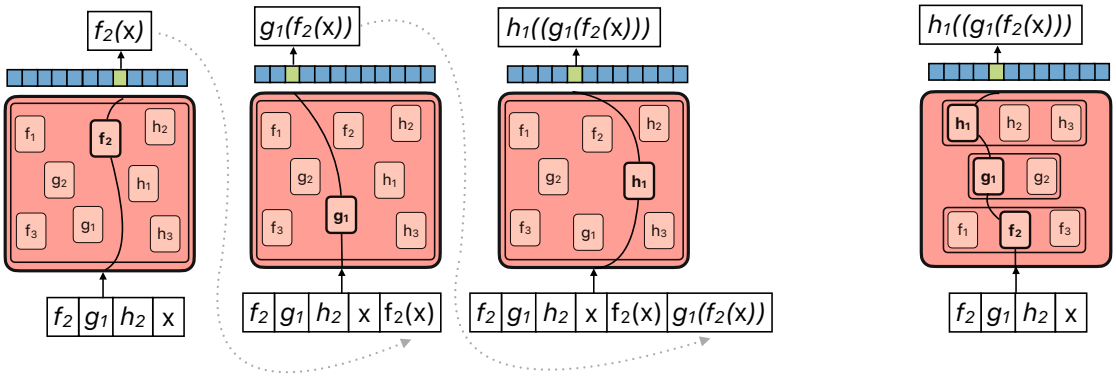


Figure 11: **Step-by-step composition v.s. Direct composition.** We test two possible routes for compositions. **(a)** Step-by-step prompting, which allows for generating intermediate outputs. **(b)** Direct prompting, where the model must compose the functions without the intermediate outputs.

Evaluating compositions When evaluating trained models, we evaluate on 1000 different inputs for every composition of functions. Since Fig. 5 requires us to evaluate on a combinatorial set of functions, we sample 1000 functions (or the

total number of functions, whichever was lower) for each cell which can be identified by the displacement and number of compositions; we then compute the accuracy averaged over those functions to populate the cell. The accuracy of a completion is calculated by averaging the accuracy of the last six tokens. We see that qualitative trends do not change when we use different metrics Fig. 19.

Computing linear probe accuracy We consider the outputs after every attention block and every MLP block (including the residual stream in both cases). We then pass these outputs through the final embedding layer and a Softmax layer to get predictions over the next token. We use these predictions to compute the accuracy at that layer. The accuracy is averaged over 1000 different input data tokens and for 200 different compositions of functions.

B. Additional Experiments

B.1. Sweeping hyper-parameters of the Transformer

We vary the number of layers, the number of attention heads, and the embedding dimension of the nanoGPT model in Fig. 13. We consider a setup identical to Fig. 4; all models are trained on 50 **random** in-order compositions of 5 bijections. We report accuracy averaged over all 3125 in-order compositions.

We make the following observations. (1) Most surprisingly, the accuracy reduces as the number of layers become *huge* for this compositional task; we expect that this is due to issues with optimization of a large depth model. (2) The accuracy does not change with the number of attention heads for a 1-layer Transformer. (3) The accuracy increases as we increase the embedding dimension and the model under fits the training data when the embedding dimension is too small.

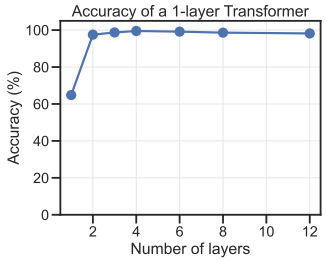


Figure 12: **Transformers requires at least 2-3 layers for compositional generalization with the direct prompt format.** We vary the number of layers in the Transformer and train on direct composition in a setup identical to Fig. 6 (Right).

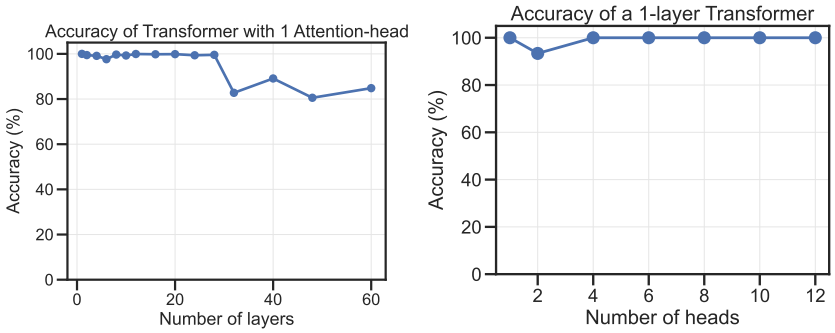


Figure 13: **We see compositionality in Transformers even if we change the number of layers and attention heads.** Compositionality is seen even in a 1-layer Transformer when trained with the step-by-step prompt format on 50 in-order compositions of bijections. However the ability to compose degrades as we increase the number of layers in the Transformer.

B.2. LSTMs do not learn to compose

We report results on autoregressively trained LSTMs using the direct prompt format from Table 1 and the step-by-step prompt format in Table 3. **LSTMs fail to generalize outside of the training data while Transformers generalize compositionally in both these scenarios**. *This points to an inductive bias that helps Transformers trained with an autoregressive objective generalize.* Specifically, our mechanistic evaluation in Sec. 4.4 shows this is likely attributable to the use of Attention.

The LSTMs are trained using the same data using the autoregressive objective defined in Appendix A. We use the AdamW optimizer with learning rate equal to 3e-4 ($\beta_1 = 0.9$ and $\beta_2 = 0.95$), batch size of 512 and weight decay of 1e-4 for 150

epochs. As is common, we do not use a positional embedding, since the architecture is not permutation invariant.

The inputs are passed through an input embedding layer before being passed to the LSTM and the outputs of the LSTM are also passed through a linear layer which outputs the logits. In our experiments, we vary the number of stacked LSTMs (or no. of layers) and the dimension of the internal hidden vector.

Layers	Hidden dimension 256	5124
1	22.5	46.0
2	33.4	69.1

Table 1: **LSTMs fail to compose in the direct prompt format.** We train an LSTM on 250 composition of two functions (one permutation and one bijection) in the direct prompt format and tabulate the accuracy (%); the setup is identical to Fig. 6 (Right).

Despite our attempt to train multiple different LSTMs with the best set of hyper-parameters, we observe that they do not show any compositional generalization on all our synthetic setups. This observation is further evidence for our hypothesis that the attention layers are important for compositionality.

One would expect that LSTMs generalize poorly since they are required to capture long range dependencies in the input in order to successfully compose functions. As a result, we consider an alternative prompt format that reduces the burden of capturing long-range dependencies. In particular, we consider a prompt of the form $x_d f_1 o_1 f_2 o_2 \dots F_L o_L$, where x_d is the input data tokens, and f_i is the i$^{\text{th}}$ task tokens while o_i is the intermediate output of the composition. In this prompt format, the relevant task token is only presented after the intermediate output is generated, as opposed to presenting all the task tokens at the start of the prompt. Using this prompt format, we trained LSTMs on compositions of 5 functions where each position of the composition can be one of 5 functions, similar to the setup of Fig. 4a. We trained on a random subset of 100 in-order compositions. We then evaluate the trained LSTMs on all in-order compositions (even ones not seen during training) and tabulate the accuracies in Table 2. LSTMs still fail to achieve perfect accuracy indicating the inductive bias of the Transformer architecture is important for compositionality. The failure of LSTMs cannot be solely attributed to the inability to capture long-range dependencies.

Layers	Hidden dimension 120	256	512	1024
1	11.64	85.42	84.83	86.61
2	11.33	86.25	88.29	88.28
4	11.44	88.52	53.11	11.33

Table 2: **LSTMs perform significantly better with the modifiied prompt format but still fail to perfectly compositionally generalize.** We train an LSTM on 100 compositions of in-order compostions with a modified prompt format that does not require capturing long-range dependencies. While the accuracies improve significantly, the LSTM still fails to compositionally generalize.

Layers	Hidden layer dimension 120	256	512	1024
1	16.2	36.2	99.9	99.9
2	60.3	99.3	99.9	99.8
4	18.7	100.0	100.0	9.9

Layers	Hidden layer dimension 120	256	512	1024
1	9.3	10.3	20.1	22.9
2	12.4	21.3	25.3	28.8
4	6.6	13.9	17.6	10.0

Table 3: **LSTMs fail to compose in the step-by-step prompt format.** We train autoregressive LSTMs on 50 in-order compositions of 5 bijections from $\mathcal{F}_b$ in the step-by-step format and tabulate the accuracy (%); The setup is identical to Fig. 4. We evaluate the LSTM on the **(left)** compositions seen during training and **(right)** in-order compositions not seen during training. LSTMs fail to generalize to functions outside of the training data while transformers generalize compositionally in the same setting.

B.3. Attention Masks

Detailed setup. We train a 1-layer Transformer on a composition of 50 **random** in-order compositions of 5 bijections in the step-by-step prompt format. We visualize the attention masks for a fixed sequence of task tokens, averaged over 1000 different data tokens in Fig. 7(right). We found the attention masks to be identical across different choices of the task tokens. Each row corresponds to a causal attention mask for a single token and sums up to 1. At any given row, the attention is over two elements—the task token and the intermediate output of the composition. The five contiguous blocks along the columns correspond to the five steps of composition. These preliminary results indicate that it is possible to build a complete mechanistic understanding of attention for compositional tasks (see also Sec. C).

B.4. Probing the layers in Transformers of different sizes

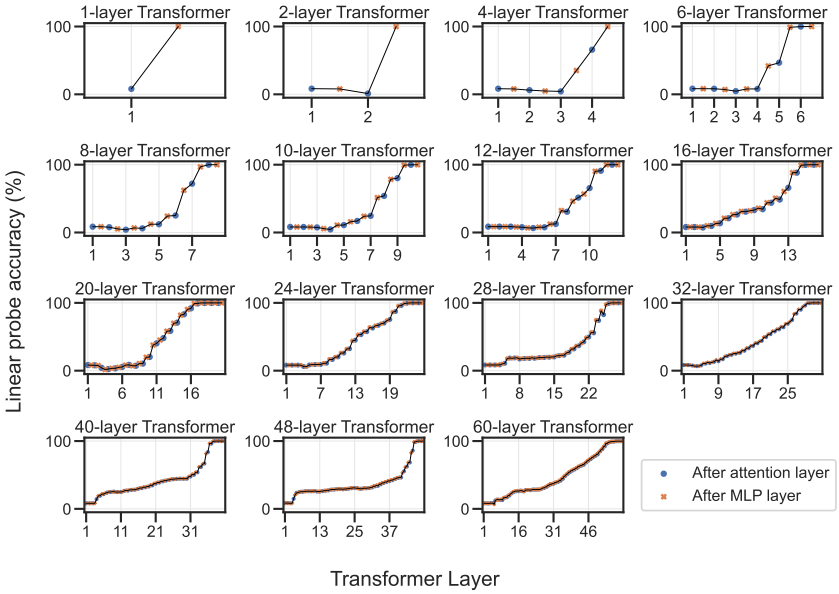


Figure 14: **We use a linear probe to study the accuracy at different layers on Transformers of different sizes.** Most architectures see an increasing in accuracy in the latter half of the Transformer. The increase in accuracy is more gradual for Transformers with more layers. The accuracy increases sharply after an attention layer across all architectures.

In this section, we consider an experimental setup that is identical to the linear probe experiments in Fig. 7. We compute the probe accuracies for Transformers with different number of layers in Fig. 14. Across all models, we observe that accuracy increases in the last few layers. Furthermore, we also observe a sharp increase in accuracy right after the MLPs in the last few layers of the transformer.

We saw in Fig. 7(right) that the attention masks for a 1-layer model seem to select an input and a task token to operate on at every step of the composition. We hence believe that attention has a huge role in compositionality and propose the following hypothesis: The probe accuracy after some MLPs see a sharp in increase in accuracy because the attention layers play a critical role in selecting the right inputs to pass to the MLP. Specifically, unlike the 1-layer model, we suspect functions are now distributed across the model layers instead of being localized in the first MLP layer. Consequently, similar to the 1-layer model, attention heads at different layers will infer if the relevant functions implemented in MLP layers in that block are part of the prompt; if so, they transfer the input data through said function.

B.5. Another failure with the direct format with bijections

In Fig. 6 (Left) we show that Transformers do not learn to compose 5 bijections and only generalize to compositions in the training data. Fig. 15 augments this result and shows that a similar failure occurs even when we consider the composition of just two bijections. Hence the model may not compose some function in the direct prompt format and the step-by-step format with an autoregressive objective is far more amenable to compositions.

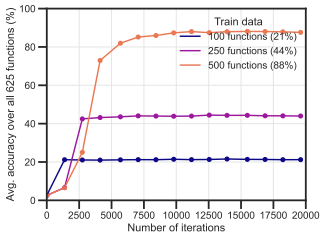


Figure 15: **Transformers fail to generalize to compositions of even 2 bijections, when trained with the direct prompt format.** The curve depicts the accuracy over all 625 in-order compositions of two bijections (25 choices for each bijection) when trained on different subsets of in-order compositions. The model is trained with direct composition. Even if we train on 500 such compositions, the model fails to generalize to the remaining 125 compositions. This is additional evidence that the model is incapable composing bijections through direct composition.

B.6. Additional experiments with training data from random and base

In this section, we conduct a collection of analyses for a model trained on in-order compositions of 5 bijections in the step-by-step prompt format. We perform the following experiments: (1) change the number of **random** functions in the training data (Fig. 16); (2) compare how **base** and **random** generalize to other in-order compositions (Fig. 17); (3) limit the maximum number of compositions in the training data and evaluate compositional generalization (Fig. 18); (4) look at alternate evaluation metrics (Fig. 19); and (5) test if the compositions are systematic (Hupkes et al., 2020) (Fig. 20).

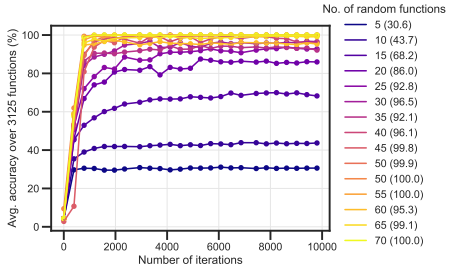


Figure 16: **Training with different numbers of random functions.** We train on a different number of random functions ranging from 5-70 in steps of 5. These plots are the accuracies averaged over all in-order compositions of 5 bijections over the course of training.

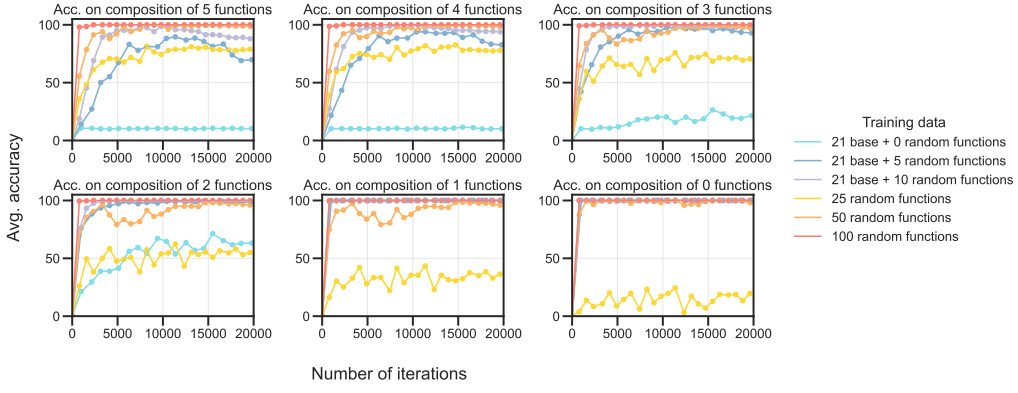


Figure 17: **How do different training datasets generalize to compositions of many and few functions?** This is a fine-grained version of Fig. 4a. Model trained on 50 **random** compositions generalizes poorly compositions of small number of functions while a model trained on the **base** generalizes poorly to composition of 4 or 5 functions.

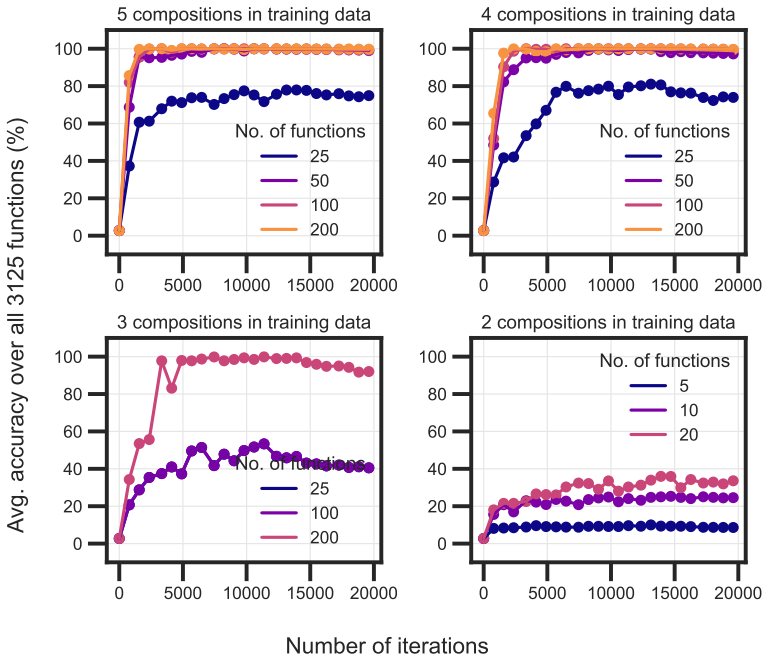


Figure 18: **Limiting maximum number of compositions in the training data.** The figure plots the accuracy on all in-order compositions against the number of training iterations. Each sub-plot considers compositions of size exactly 2, 3, 4, 5, respectively in the training data. The model is able to generalize to most in-order compositions only if the training data consists of compositions of size at least 3 (bottom-right).

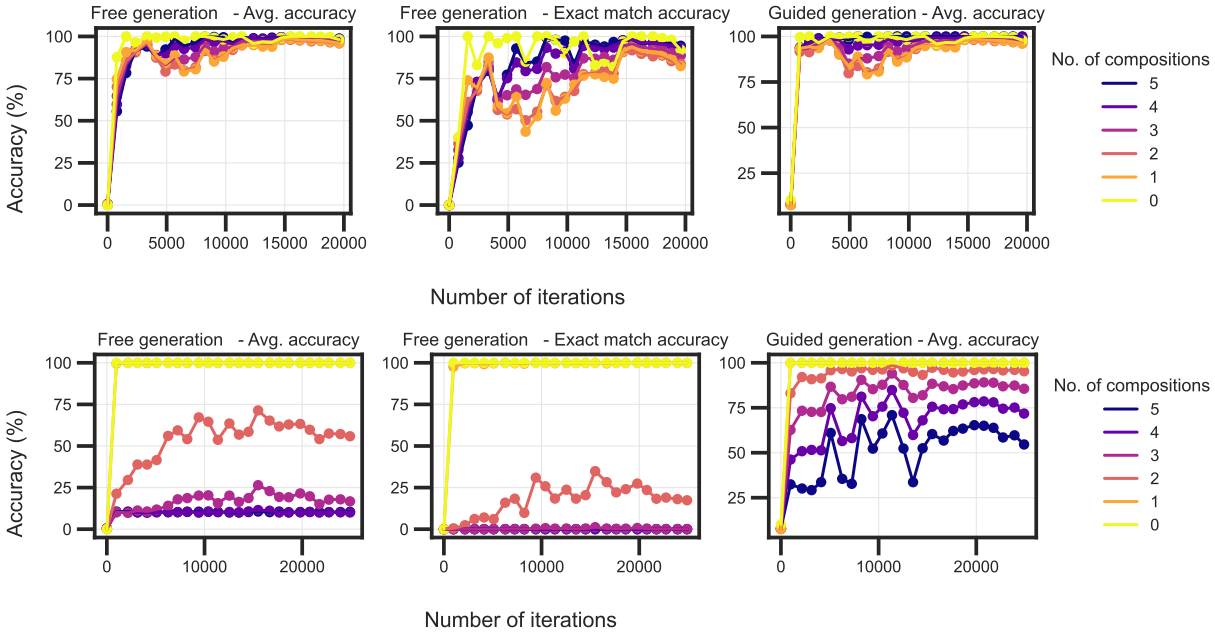


Figure 19: **Evaluation metric.** We consider 3 different metrics for evaluating the models. The left column considers the average accuracy when the model generates **The choice of metric doesn't change qualitative trends.** Each sub-plot considers compositions of only size 2, 3, 4, 5, respectively. In each plot, we vary the number of such functions that are present int he training data. **One exception is when we train on compositions of size 2.** In this case, the guided generation accuracy is high, but the free generation accuracy is not.

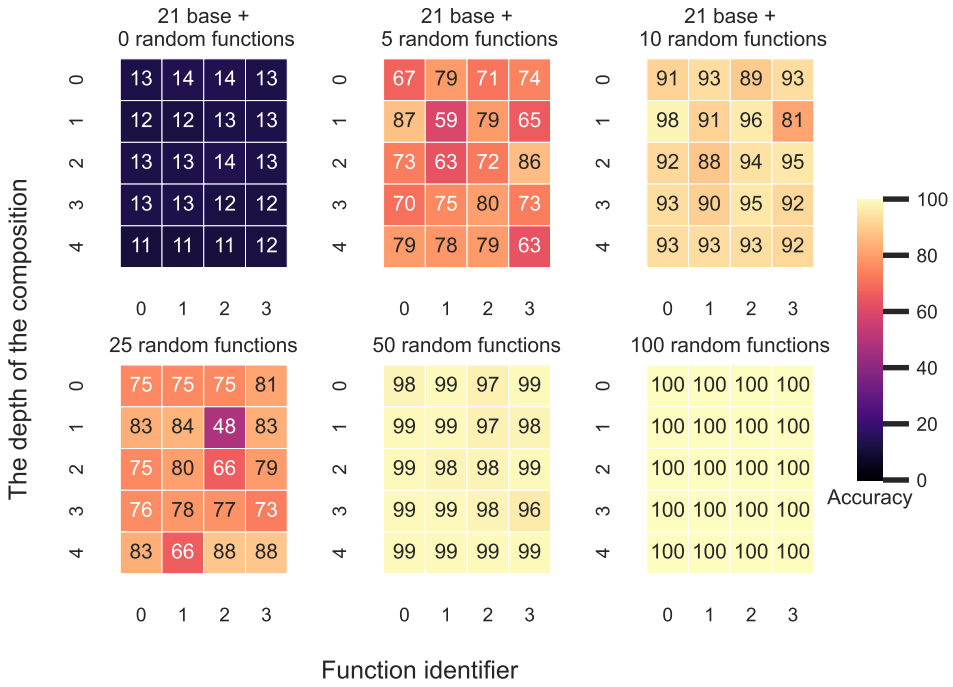


Figure 20: **Systematicity.** We consider trained models from Fig. 4a and analyze the accuracy of each of the 20 functions (atomic capabilities) when averaged all instances in which it was used compositionally. We breakdown the results to see if certain functions are more accurate when used in compositions compared to others and find that models seem to learn all functions equally well.

B.7. Token embeddings

We study the token embeddings of the Transformer models and observe that they are similar for models with different number of layers and attention heads (see Fig. 21). We notice a block diagonal structure that separates task tokens from the data tokens. We also observe another block diagonal structure within the task tokens which occurs when we train only on in-order compositions.

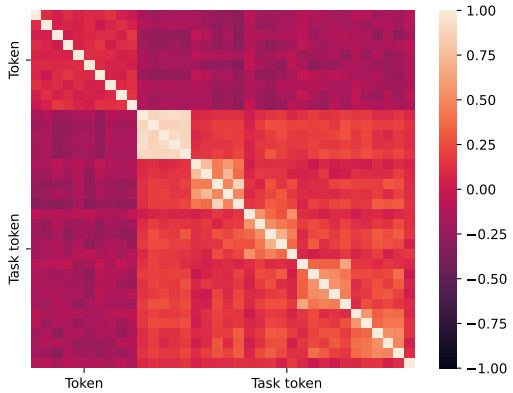


Figure 21: **Word embedding correlations present a block-diagonal structure that separates data tokens from task tokens.** We plot the inner product between all pairs of word embeddings of the tokens. The task tokens are orthogonal to the set of input tokens. Different functions in the same level, i.e. $\{F_i^{(l)}\}_{i=1}^{N}$ for a fixed l, form a block-diagonal in this matrix. We observe similar word embeddings in Transformers of different sizes.

C. Analysis of Step-by-step and Direct Prompt Formats

C.1. Transformers for the step-by-step prompt format

We prove that there exists Transformers that can compositionally generalize in the step-by-step prompt format. Such a constructive proof, similar to (Von Oswald et al., 2023; Ahn et al., 2023; Weiss et al., 2021; Li et al., 2023c), can be used to generate plausible mechanistic hypothesis by highlighting the role of the attention and MLP layers. While the universal approximation theorem suggests that any function can be represented by a wide enough multi-layer perceptron (MLP), the construction suggests that Transformers can represent the same function efficiently.

Description of the data. We will operate with a simplified prompt format where a composition of three functions is to be applied to a single input token. The construction can be generalized to compositions of more functions or to multiple input tokens. The input prompt $[x_{F_1}, x_{F_2}, x_{F_3}, x_d]$ has three task tokens and a single data token, and the desired output for this prompt is $[F_1(x_d), F_2 \circ F_1(x_d), F_3 \circ F_2 \circ F_1(x_d)]$.

The position encodings $P = \begin{bmatrix} p_1 & p_2 & \cdots & p_6 \end{bmatrix}$ are learnable parameters and have dimension d_p, i.e., $P \in \mathbb{R}^{d_p \times 6}$. The number of input tokens is d_v and the number of task tokens is d_f. Both input tokens x_d and task tokens x_{F_1} are embedded as a one-hot vector in $\mathbb{R}^{d_x}$ where $d_x = d_v + d_f$. The first d_v dimensions are used to embed the data tokens and the last d_f dimensions embed the task token. Henceforth, both x_d and x_{F_1} refer to the corresponding one-hot vectors in $\mathbb{R}^{d_x}$. For convenience, we define $d = d_x + d_p$. Tying this back to to section 3, observe that $|X_d| = d_v$ and $|X_f| = d_f$. We denote the input to the model using Z, which includes the token embedding and position encoding. Specifically, we have

$$Z = \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & x & F_1(x_d) & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{bmatrix},$$

i.e., $Z \in \mathbb{R}^{d \times 6}$. We assume that the position encoding is concatenated to the token embedding as opposed to added to it.

Matrix notation. We use $\mathbb{1}_{x_d}$ to denote a one-hot vector in the space $\mathbb{R}^{d_v}$, i.e., it excludes dimensions for the task token. On the other hand, x_d denotes a one-hot vector in $\mathbb{R}^{d_x}$. We use $I_{n\times n}$ to denote an identity matrix of size $n \times n$, $1_{m\times n}$ and $0_{m\times n}$ to denote matrices of 1s and 0s of size $m \times n$, and 1_n and 0_n to denote matrices of size $n \times 1$.

Description of the architecture. Before describing the Transformer architecture, we first define the attention and MLP layers. We use a simplified parameterization of linear attention (Ahn et al., 2023) with weights Q and K. The MLP contains two fully connected layers with a ReLU non-linearity parameterized by the weights W_1 and W_2. The attention and MLP layers are functions of $Z \in \mathbb{R}^{d\times 6}$ and are defined as:

$$\begin{aligned} &\mathrm{Attn}_{Q,K}(Z) = (KZ)(M \odot Z^T QZ), \text{ and} \\ &\mathrm{MLP}_{W_1,W_2}(Z) = W_2 \mathrm{ReLU}(W_1 Z), \end{aligned}$$

where $Q, K \in \mathbb{R}^{d\times d}$, $W_1 \in \mathbb{R}^{d\times(d_f d_v)}$ and $W_2 \in \mathbb{R}^{(d_f d_v)\times d}$. The matrix $M \in \mathbb{R}^{6\times 6}$ enforces causal attention and restricts the attention to inputs from previous time-steps, i.e.,

$$M = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

We consider a 1-layer Transformer with an attention layer followed by an MLP layer. We omit layer-norm to simplify the proofs. The function computed by the Transformer is

$$\mathrm{Tr}_{Q,K,W_1,W_2}(Z) = \mathrm{MLP}\left(\mathrm{Attn}(Z) + Z\right) + \mathrm{Attn}(Z) + Z\,.$$

Henceforth, we omit the subscripts of Attn, MLP and Tr for brevity. We include a residual connection after both the attention and MLP layers which mirrors a typical Transformer architecture (Vaswani et al., 2017).

The output of the Transformer is passed through an unembedding matrix W_e followed by a Softmax layer to obtain a probability distribution over the next token denoted by

$$P(Y|Z) = \mathrm{Softmax}(W_e \mathrm{Tr}(Z)).$$

Theorem C.1. *There exists weights P, Q, K, W_1, W_2 and position encodings P such that an Autoregressive Transformer can compositionally generalize to any prompt $[x_{F_1}, x_{F_2}, x_{F_3}, x_d]$. The values of the weights satisfy*

$$P^T P = \begin{bmatrix} I_{3\times 3} & I_{3\times 3} \\ I_{3\times 3} & I_{3\times 3} \end{bmatrix}, \qquad Q = \begin{bmatrix} 0_{d\times d} & 0_{d\times d_p} \\ 0_{d_p\times d} & I_{d_p\times d_p} \end{bmatrix}, \qquad K = \begin{bmatrix} 0_{d_v\times d_v} & 0_{d_f\times d_v} & 0_{d\times d_p} \\ 0_{d_f\times d_v} & I_{d_f\times d_f} & 0_{d\times d_p} \\ 0_{d_v\times d} & 0_{d_f\times d} & 0_{d_p\times d_p} \end{bmatrix},$$

$$W_1 = \underbrace{\begin{bmatrix} \mathbb{1}^T_{x_{d_1}} & \mathbb{1}^T_{x_{d_1}} & \mathbb{1}^T_{x_{d_1}} & \cdots & \mathbb{1}^T_{x_{d_1}} \\ \mathbb{1}^T_{x_{d_2}} & \mathbb{1}^T_{x_{d_2}} & \mathbb{1}^T_{x_{d_2}} & \cdots & \mathbb{1}^T_{x_{d_2}} \\ \vdots & \vdots & \vdots & \vdots & \\ \mathbb{1}^T_{x_{d_v}} & \mathbb{1}^T_{x_{d_v}} & \mathbb{1}^T_{x_{d_v}} & \cdots & \mathbb{1}^T_{x_{d_v}} \\ 0^T_{d_v} & -1^T_{d_v} & -1^T_{d_v} & \cdots & -1^T_{d_v} \\ -1^T_{d_v} & 0^T_{d_v} & -1^T_{d_v} & \cdots & -1^T_{d_v} \\ \vdots & \vdots & \vdots & \vdots & \\ -1^T_{d_v} & -1^T_{d_v} & -1^T_{d_v} & \cdots & 0^T_{d_v} \\ 0_{d_p\times d_v} & 0_{d_p\times d_v} & 0_{d_p\times d_v} & \cdots & 0_{d_p\times d_v} \end{bmatrix}^T}_{d_f\times d_v \text{columns}}, \text{ and } \quad W_2 = \begin{bmatrix} F_{i_1}(x_{d_1})^T - x^T_{d_1} - x^T_{F_{i_1}} \\ F_{i_1}(x_{d_2})^T - x^T_{d_2} - x^T_{F_{i_1}} \\ \vdots \\ F_{i_1}(x_{d_v})^T - x^T_{d_v} - x^T_{F_{i_1}} \\ F_{i_2}(x_{d_1})^T - x^T_{d_1} - x^T_{F_{i_2}} \\ F_{i_2}(x_{d_2})^T - x^T_{d_2} - x^T_{F_{i_2}} \\ \vdots \\ F_{i_2}(x_{d_v})^T - x^T_{d_v} - x^T_{F_{i_2}} \\ \vdots \\ F_{i_T}(x_{d_1})^T - x^T_{d_1} - x^T_{F_{i_T}} \\ F_{i_T}(x_{d_2})^T - x^T_{d_2} - x^T_{F_{i_T}} \\ \vdots \\ F_T(x_{d_v}) - x_{d_v} - x_{F_{i_T}} \end{bmatrix}^T.$$

Proof. See Appendix C.4. □

The construction uses the attention layer to aggregate the task token and data token, i.e., attention selects the relevant task token. The query vector of the attention selects the right task using the position encoding. The first layer of the MLP projects the summation of the task and data tokens (present in orthogonal spaces) onto the Cartesian product of the set of task and data tokens. The second layer computes the function and acts similar to a lookup table (Geva et al., 2022).

The construction requires the output of the first fully-connected layer has size at least $d_f d_v$ in order to encode the task and input tokens. In our experiments, we set $d_v = 10$ and $d_f = 21$ and hence the number of hidden units must be at least 210. In practice, we require at least 500 hidden units (see Fig. 22), which is not too far from our estimate. We conjecture that the additional hidden units are helpful for optimization.

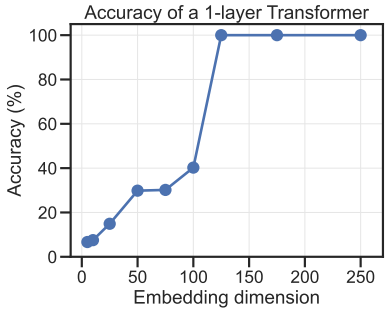


Figure 22: We see a sharp increase in accuracy as we increase the embedding dimension of the Transformer. The number of hidden units in the MLP of the Transformer is 4 times the size of the embedding dimension.

C.2. Transformers for the direct prompt format

We also prove the existence of a Transformer for a compositions of bijections in the direct prompt format. Unlike the step-by-step format, the direct prompt format lacks a "scratchpad" (Nye et al., 2021) for the intermediates outputs of the composition. In our construction, we use $K = 3$ Transformer blocks to compute the composition of K functions; the output of the k-th block is the result of the k^{th} step of the composition.

Description of the data. We consider the composition of 3 functions with an input prompt denoted by $[x_{F_1}, x_{F_2}, x_{F_3}, x_d]$. Unlike the step-by-step format, the output is just a single token $[F_3 \circ F_2 \circ F_1(x_d)]$. The position encodings are denoted by $P = [p_1, p_2, \ldots, p_4]$ where $p_i = \begin{bmatrix} p^T_{i1} & p^T_{i2} & p^T_{i3} \end{bmatrix}^T$ and $p_i \in \mathbb{R}^{d_p}$ and $p_{ij} \in \mathbb{R}_{d_p/3}$. The dimensions d_x, d_v, d and d_p

represent the same quantities. We use $\bar{d}_p$ to replace $\frac{d_p}{3}$. The input to the model is

$$Z = \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & x_d \\ p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix},$$

where $Z \in \mathbb{R}^{d \times 4}$.

Description of the architecture. Each Transformer block is defined similar to the step-by-step format, i.e.,

$$\text{Block}_{Q_i,K_i,W_{i1},W_{i2}}(Z) = \text{MLP}_i(\text{Attn}_i(Z) + Z) + (\text{Attn}_i(Z) + Z),$$

which we henceforth denote by $\text{Block}_i(Z)$. Unlike the step-by-step format, the model is now composed of 3 blocks corresponding to the 3 steps of the compositional task the model is expected to solve, i.e.,

$$\text{Tr}(Z) = \text{Block}_3(\text{Block}_2(\text{Block}_1(Z))).$$

This input is passed through a Softmax layer to predict a probability distribution over the next token, denoted by $P(Y \mid Z) = \text{Softmax}(W_e \text{Tr}(Z))$.

Theorem C.2. *There exist weights $P_i, Q_i, K_i, W_{1i}, W_{2i}$ for $i \in [1, 3]$ and position encodings P such that the a 3-layer Transformer can compositionally generalize to any prompt of the form $[x_{F_1}, x_{F_2}, x_{F_3}, x_d]$. The values of the weights satisfy*

$$Q_1 = \begin{bmatrix} 0_{d\times d} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & I_{\bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \end{bmatrix}, \qquad Q_2 = \begin{bmatrix} 0_{d\times d} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & I_{\bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \end{bmatrix},$$

$$Q_3 = \begin{bmatrix} 0_{d\times d} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & I_{\bar{d}_p}, \end{bmatrix}, \qquad K_1 = \begin{bmatrix} 0_{d_v\times d_v} & 0_{d_f\times d_v} & 0_{d\times d_p} \\ 0_{d_f\times d_v} & I_{d_f} & 0_{d\times d_p} \\ 0_{d_v\times d} & 0_{d_f\times d} & 0_{d_p\times d_p} \end{bmatrix},$$

$$K_2 = \frac{K_1}{2}, \qquad K_3 = \frac{K_1}{3},$$

$$P_1^T P_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \qquad P_2^T P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \qquad P_3^T P_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

$$W_{11} = \underbrace{\begin{bmatrix} \mathbb{1}_{x_{d_1}}^T & \mathbb{1}_{x_{d_1}}^T & \mathbb{1}_{x_{d_1}}^T & \cdots & \mathbb{1}_{x_{d_1}}^T \\ \mathbb{1}_{x_{d_2}}^T & \mathbb{1}_{x_{d_2}}^T & \mathbb{1}_{x_{d_2}}^T & \cdots & \mathbb{1}_{x_{d_2}}^T \\ \vdots & \vdots & \vdots & \vdots & \\ \mathbb{1}_{x_{d_v}}^T & \mathbb{1}_{x_{d_v}}^T & \mathbb{1}_{x_{d_v}}^T & \cdots & \mathbb{1}_{x_{d_v}}^T \\ 0_{1\times d_v} & -1_{1\times d_v} & -1_{1\times d_v} & \cdots & -1_{1\times d_v} \\ -1_{1\times d_v} & 0_{1\times d_v} & -1_{1\times d_v} & \cdots & -1_{1\times d_v} \\ \vdots & \vdots & \vdots & \vdots & \\ -1_{1\times d_v} & -1_{1\times d_v} & -1_{1\times d_v} & \cdots & 0_{1\times d_v} \\ 0_{d_p\times d_v} & 0_{d_p\times d_v} & 0_{d_p\times d_v} & \cdots & 0_{d_p\times d_v} \end{bmatrix}^T}_{d_f \times d_v \text{columns}}, \quad W_{12} = \begin{bmatrix} F_{i_1}(x_{d_1})^T - x_{d_1}^T - x_{F_{i_1}}^T \\ F_{i_1}(x_{d_2})^T - x_{d_2}^T - x_{F_{i_1}}^T \\ \vdots \\ F_{i_1}(x_{d_v})^T - x_{d_v}^T - x_{F_{i_1}}^T \\ F_{i_2}(x_{d_1})^T - x_{d_1}^T - x_{F_{i_2}}^T \\ F_{i_2}(x_{d_2})^T - x_{d_2}^T - x_{F_{i_2}}^T \\ \vdots \\ F_{i_2}(x_{d_v})^T - x_{d_v}^T - x_{F_{i_2}}^T \\ \vdots \\ F_{i_T}(x_{d_1})^T - x_{d_1}^T - x_{F_{i_T}}^T \\ F_{i_T}(x_{d_2})^T - x_{d_2}^T - x_{F_{i_T}}^T \\ \vdots \\ F_T(x_{d_v}) - x_{d_v} - x_{F_{i_T}}. \end{bmatrix}^T$$

$$W_{21} = W_{22} = W_{23}, \textit{ and } \quad W_{31} = W_{32} = W_{33}.$$

Proof. See Appendix C.5. □

C.3. Difference between the direct and step-by-step prompt formats

The ability to run multiple forward passes through the Transformer allows us tackle a richer class of problems (Merrill & Sabharwal, 2023). This ability differentiates the step-by-step and direct prompt formats. In the step-by-step prompt format, the Transformer makes L different forward passes, while the direct prompt format allows only one forward pass through the model to generate the output. This is also mirrored in our constructions in appendices C.1 and C.2—a model for the step-by-step prompt format requires only 1 layer, while one for the direct prompt format uses $L = 3$ layers to compensate for the lack of multiple forward passes. We expect that a Transformer for the direct prompt format cannot circumvent these computations and conjecture that our Transformer construction for the direct format (in appendix C.5) is efficient with respect to the number of layers.

Conjecture C.3. We conjecture that a Transformer with width of $\text{poly}(|\mathcal{F}|)$, needs $\mathcal{O}(L)$ layers in the direct prompt format compared to the $\mathcal{O}(1)$ layers step-by-step format in order to compositionally generalize on our synthetic task.

That is, a model must compute all L intermediate outputs of the composition across different layers of the Transformer. We expand on this further in the next subsection. We also note that as per the universal approximation theorem, it is certainly possible to construct a Transformer with 1-layer such that it generalizes for the direct prompt format; *however, such a model must have its width to be exponential in $|\mathcal{F}|$ in order to store $|\mathcal{F}|^L$ different functions in a single layer.*

C.3.1. How many training compositions does each prompt format need?

To further understand the difference between the two prompt formats, we will use a (highly simplified) model to reason about the number of function compositions in the training data that is required for perfect compositional generalization on our task. Let us consider a composition of L of functions from $\mathcal{F}$. We assume that the compositions in the training data $\mathcal{F}_{\text{train}}^L \subset \mathcal{F}^L$ are sampled uniformly at random from the set of all compositions.

For this analysis, we assume that the Transformer can perfectly identify which functions to compose—which we ascribe to the attention layers—and will focus entirely on capability acquisition which we hypothesize is carried out by the MLP layers. We assume that a Transformer for the step-by-step prompt format must learn a function (capability) only once, while a Transformer for the direct prompt format must learn the function L different times—once for each layer of the Transformer. If the function composition $F^{(l)} \in \mathcal{F}_{\text{train}}^L$ occurs in the training data, we assume that the Transformer for the step-by-step format has learned all the capabilities $F_i^{(l)} \in F^{(l)}$ for $i \in [1, L]$, while a Transformer for the direct prompt format can only learn capability $F_i^{(l)}$ at layer i. These assumptions are informed by Theorems C.1 and C.2.

Detour into the coupon collector's problem. In order to learn all $F = |\mathcal{F}|$ capabilities, the training data must contain each capability at least once. We note that this is the coupon collector's problem (Myers & Wilf, 2006): the collector seeks all distinct coupons and recieves a coupon at every round drawn uniformly at random. The number of rounds corresponds to the number of function compositions in the training data and we would like to calculate the expected number of rounds required to learn all capabilities. It is a well known result that the expected number of rounds to collect all F coupons is FH_F where H_F is the Harmonic number; asymptotically this is $\mathcal{O}(F \log F)$. Furthermore, the probability that we complete a collection of size f, in n rounds is

$$p(L, f) = \frac{F!}{F^L} \left\{ {F-1 \atop L-1} \right\},$$

where $\left\{ {F-1 \atop K-1} \right\}$ is the Stirling number of the second kind.

In the step-by-step prompt format, we observe L capabilities (or coupons) with every composition. All capabilities are learned if we observe each of them in at least one training sample. The expected number of training compositions N required to learn all capabilities is $\mathcal{O}(\frac{F \log F}{L})$ (see Xu & Tang (2011)). On the other hand, the direct prompt format can be treated as L independent coupon collector problems and must observe each capability once for each of the L layers. The expected number of rounds to learn all capabilities is the is the expected value of maximum number of rounds for L indepedent coupon collector problems. If we apply Chebyshev's inequality, we get

$$P(N \geq FH_F + c \log F) \leq \frac{\pi^2}{6c^2 \log^2 F},$$

since the variance of N is upper bounded by $\frac{n^2\pi^2}{6}$. Hence, the maximum value of L different runs is $\mathcal{O}(F \log F)$ as $n \to \infty$, or in other words, the expected number of rounds to learn all capabilities is $\mathcal{O}(F \log F)$. The expected number of training compositions differ by a factor of L between the two prompt formats, which tallies with the observation that a Transformer is expected to learn the same set of capabilities L different times in the direct format.

In practice, we find that Transformers for the direct format can sometimes fail to compositionally generalize, even with a large number of compositions in the training data (Section 4.3). We hypothesize that this is attributable to the optimization landscape, i.e., gradient descent is unable to find weights that compositionally generalize and instead prefers weights that memorize compositions of functions present in the training data. In the direct prompt, gradient descent must recover the individual capabilities from a set of compositions of bijections and this is a computationally hard problem since it is similar to finding the minimal generating set of a group (its time complexity is linear in the size of the group which is $\mathcal{O}(F^L)$).

C.4. Proof of Theorem C.1

Step 1: Computing the attention layer. The attention layer copies the task tokens onto the relevant data token similar to an induction head (Olsson et al., 2022). We first compute the query and value matrices of the attention.

$$Z^T Q Z = \begin{bmatrix} x_{F_1}^T & p_1^T \\ x_{F_2}^T & p_2^T \\ x_{F_3}^T & p_3^T \\ x_d^T & p_4^T \\ F_1(x_d)^T & p_5 \\ F_2 \circ F_1(x_d)^T & p_6^T \end{bmatrix} \begin{bmatrix} 0_{d \times d} & 0_{d \times d_p} \\ 0_{d_p \times d} & I_{d_p \times d_p} \end{bmatrix} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & \cdots & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & \cdots & p_6 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & p_1^T \\ 0 & p_2^T \\ 0 & p_3^T \\ 0 & p_4^T \\ 0 & p_5^T \\ 0 & p_6^T \end{bmatrix} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & \cdots & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & \cdots & p_6 \end{bmatrix} = P^T P$$

Our construction considers a P such that $p_i = p_{i+4}$ for all $i \in [1, 3]$ and $p_i \cdot p_j = 0$ for all $j \in [1, 3]$ and $j \neq i$. The mask M converts $P^T P$ into an upper triangular matrix, and zeroes out all entries in the lower triangle of the matrix.

$$M \odot (Z^T Q Z) = M \odot (P^T P) = M \odot \begin{bmatrix} I_{3\times3} & I_{3\times3} \\ I_{3\times3} & I_{3\times3} \end{bmatrix} = \begin{bmatrix} I_{3\times3} & I_{3\times3} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix}$$

The attention layer computes

$$\begin{aligned}
\text{Attn}(Z) &= (KZ)(M \odot Z^T QZ) \\
&= (KZ)(M \odot PP^T) \\
&= \begin{bmatrix} 0_{d_v \times d_v} & 0_{d_f \times d_v} & 0_{d \times d_p} \\ 0_{d_f \times d_v} & I_{d_f \times d_f} & 0_{d \times d_p} \\ 0_{d_v \times d} & 0_{d_f \times d} & 0_{d_p \times d_p} \end{bmatrix} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & \cdots & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & \cdots & p_6 \end{bmatrix} \begin{bmatrix} I_{3\times 3} & I_{3\times 3} \\ 0_{3\times 3} & I_{3\times 3} \end{bmatrix} \\
&= \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0_d & 0_d & 0_d \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix} \begin{bmatrix} I_{3\times 3} & I_{3\times 3} \\ 0_{3\times 3} & I_{3\times 3} \end{bmatrix} \\
&= \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & x_{F_1} & x_{F_2} & x_{F_3} \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix}
\end{aligned}$$

which when added to Z yields

$$\text{Attn}(Z) + Z = \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & x_d + x_{F_1} & F_1(x_d) + x_{F_2} & F_2 \circ F_1(x_d) + x_{F_3} \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6, \end{bmatrix}$$

if we also include the residual stream to the output of the attention layer.

Step 2: Computing the MLP layer. After the attention layer, the data and task tokens are aggregated at one location in orthogonal sub-spaces. The MLP uses the task and data token to compute the function. The first fully-connected layer projects the input $\mathbb{R}_{d_v d_t}$, which uniquely identifies the task and data tokens which is used to retrived the function from W_2. The first fully-connected layer computes

$$\begin{aligned}
(\text{Attn}(Z) + Z)^T W_1^T &= \begin{bmatrix} 2x_{F_1}^T & p_1^T \\ 2x_{F_2}^T & p_2^T \\ 2x_{F_3}^T & p_3^T \\ x_d^T + x_{F_1}^T & p_4^T \\ F_1(x_d)^T + x_{F_2}^T & p_5^T \\ F_2(F_1(x_d))^T + x_{F_3}^T & p_6^T \end{bmatrix} \begin{bmatrix} \mathbb{1}_{x_{d_1}}^T & \mathbb{1}_{x_{d_1}}^T & \mathbb{1}_{x_{d_1}}^T & \cdots & \mathbb{1}_{x_{d_1}}^T \\ \mathbb{1}_{x_{d_2}}^T & \mathbb{1}_{x_{d_2}}^T & \mathbb{1}_{x_{d_2}}^T & \cdots & \mathbb{1}_{x_{d_2}}^T \\ \vdots & \vdots & \ddots & \vdots & \\ \mathbb{1}_{x_{d_v}}^T & \mathbb{1}_{x_{d_v}}^T & \mathbb{1}_{x_{d_v}}^T & \cdots & \mathbb{1}_{x_{d_v}}^T \\ 0_{d_v}^T & -1_{d_v}^T & -1_{d_v}^T & \cdots & -1_{d_v}^T \\ -1_{d_v}^T & 0_{d_v}^T & -1_{d_v}^T & \cdots & -1_{d_v}^T \\ \vdots & \vdots & \ddots & \vdots & \\ -1_{d_v}^T & -1_{d_v}^T & -1_{d_v}^T & \cdots & 0_{d_v}^T \\ 0_{d_p \times d_v} & 0_{d_p \times d_v} & 0_{d_p \times d_v} & \cdots & 0_{d_p \times d_v} \end{bmatrix} \\
&= \begin{bmatrix} -2_{d_v}^T & \cdots & \cdots & 0_{d_v}^T & \cdots & \cdots & -2_{d_v}^T \\ -2_{d_v}^T & \cdots & 0_{d_v}^T & \cdots & \cdots & \cdots & -2_{d_v}^T \\ -2_{d_v}^T & \cdots & \cdots & \cdots & 0_{d_v}^T & \cdots & -2_{d_v}^T \\ -1_{d_v}^T + \mathbb{1}_{x_d}^T & \cdots & \cdots & \mathbb{1}_{x_d}^T & \cdots & \cdots & -1_{d_v}^T + \mathbb{1}_{x_d}^T \\ -1_{d_v}^T + \mathbb{1}_{F_1(x_d)}^T & \cdots & \mathbb{1}_{F_1(x_d)}^T & \cdots & \cdots & \cdots & -1_{d_v}^T + \mathbb{1}_{F_1(x_d)}^T \\ -1_{d_v}^T + \mathbb{1}_{F_2 \circ F_1(x_d)}^T & \cdots & & \cdots & \mathbb{1}_{F_2 \circ F_1(x_d)}^T & \cdots & -1_{d_v}^T + \mathbb{1}_{F_2 \circ F_1(x_d)}^T \end{bmatrix}
\end{aligned}$$

The above matrix has $d_f d_v$ columns represented as d_f blocks of size d_v. The 0 matrix in the first, second and third row occupy d_v columns each. In particular, they occupy the blocks j_1, j_2 and j_3 where $F_i = F_{i_{j_i}}$, i.e. the block number corresponds to index in the one-hot representation of the task tokens. Let $\mathbb{1}_{(x,F)}$ denote a one-hot vector in $\mathbb{R}^{d_v \times d_f}$, i.e., it is a one-hot vector that uniquely identifies the task and data token. We can succincintly express the output after the

non-linearity as follows:

$$\begin{aligned}\mathrm{ReLU}(W_1(\mathrm{Attn}(Z)+Z)) &= \mathrm{ReLU}((\mathrm{Attn}(Z)+Z)^T W_1^T)^T) \\ &= \begin{bmatrix} 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} \\ 0_{d_v} & 0_{d_v} & 0_{d_v} & \cdots & \cdots & \cdots \\ 0_{d_v} & 0_{d_v} & 0_{d_v} & \cdots & \mathbb{1}_{F_1(x_d)} & \cdots \\ 0_{d_v} & 0_{d_v} & 0_{d_v} & \mathbb{1}_{x_d} & \cdots & \cdots \\ 0_{d_v} & 0_{d_v} & 0_{d_v} & \cdots & \cdots & \mathbb{1}_{F_2\circ F_1(x_d)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} & 0_{d_v} \end{bmatrix} \\ &= \begin{bmatrix} 0_{d_v d_f} & 0_{d_v d_f} & 0_{d_v d_f} & \mathbb{1}_{(x_d,F_1)} & \mathbb{1}_{(F_1(x_d),F_2)} & \mathbb{1}_{(F_2\circ F_1(x_d),F_3)} \end{bmatrix}\end{aligned}$$

Including the final weight matrix W_2, we get

$$\begin{aligned}W_2\mathrm{ReLU}(W_1(\mathrm{Attn}(Z)+Z)) &= W_2 \begin{bmatrix} 0_{d_v d_f} & 0_{d_v d_f} & 0_{d_v d_f} & \mathbb{1}_{(x_d,F_1)} & \mathbb{1}_{(F_1(x_d),F_2)} & \mathbb{1}_{(F_2\circ F_1(x_d),F_3)} \end{bmatrix} \\ &= \begin{bmatrix} 0_d^T & 0_{d_p}^T \\ 0_d^T & 0_{d_p}^T \\ 0_d^T & 0_{d_p}^T \\ F_1(x_d)^T - x_d - x_{F_1} & 0_{d_p}^T \\ F_2\circ F_1(x_d) - x_{F_1(x_d)} - x_{F_2} & 0_{d_p}^T \\ F_3\circ F_2\circ F_1(x_d) - x_{F_2\circ F_1(x_d)} - x_{F_3} & 0_{d_p}^T \end{bmatrix}^T\end{aligned}$$

Hence, the output of the Transformer is

$$\begin{aligned}\mathrm{Tr}(Z) &= \mathrm{MLP}(\mathrm{Attn}(Z)+Z) + (\mathrm{Attn}(Z)+Z) \\ &= \begin{bmatrix} 0_d^T & 0_{d_p}^T \\ 0_d^T & 0_{d_p}^T \\ 0_d^T & 0_{d_p}^T \\ F_1(x_d)^T - x_d^T - x_{F_1}^T & 0_{d_p}^T \\ F_2\circ F_1(x_d)^T - x_{F_1(x_d)}^T - x_{F_2}^T & 0_{d_p}^T \\ F_3\circ F_2\circ F_1(x_d)^T - x_{F_2\circ F_1(x_d)}^T - x_{F_3}^T & 0_{d_p}^T \end{bmatrix}^T + \begin{bmatrix} 2x_{F_1}^T & p_1^T \\ 2x_{F_2}^T & p_2^T \\ 2x_{F_3}^T & p_3^T \\ x_d^T + x_{F_1}^T & p_4^T \\ x_{F_1(x_d)}^T + x_{F_2}^T & p_5^T \\ x_{F_2\circ F_1(x_d)}^T + x_{F_3}^T & p_6^T \end{bmatrix}^T \\ &= \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & F_1(x_d) & F_2\circ F_1(x_d) & F_3\circ F_2\circ F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{bmatrix}\end{aligned} \tag{0}$$

If we set

$$W_e = \begin{bmatrix} I_{d\times d} & 0_{d\times d_p} \\ 0_{d_p\times d} & o_{d_p\times d_p} \end{bmatrix},$$

then $W_e\mathrm{Tr}(Z)$ evaluates to

$$\begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & F_1(x_d) & F_2\circ F_1(x_d) & F_3\circ F_2\circ F_1(x_d) \end{bmatrix}$$

which will assign high probabilities to the desired token when passed through a Softmax layer. Hence, a Transformer prompted with $[x_{F_1}, x_{F_2}, x_{F_3}, x_d]$ will auto-regressively generate $[F_1(x_d), F_2\circ F_1(x_d), F_3\circ F_2\circ F_1(x_d)]$ for any combination of data and task tokens. □

C.5. Proof of Theorem C.2

The details of construction are similar to Appendix C.4.

Step 1: Computing the output of the first block. The first Transformer block computes the first step of the composition. The attention layer in particular, copies the relevant task token to the data token. The value and query matrices of the

attention layer in the first Transformer block are

$$\begin{aligned} Z^T Q_1 Z &= \begin{bmatrix} x_{F_1}^T & p_{11}^T & p_{21}^T & p_{31}^T \\ x_{F_2}^T & p_{12}^T & p_{22}^T & p_{32}^T \\ x_{F_3}^T & p_{13}^T & p_{23}^T & p_{33}^T \\ x_d^T & p_{14}^T & p_{24}^T & p_{34}^T \end{bmatrix} \begin{bmatrix} 0_{d\times d} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & I_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \end{bmatrix} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & x_d \\ p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \\ &= P_1^T P_1, \end{aligned}$$

and

$$K_1 Z = \begin{bmatrix} 0_{d_v\times d_v} & 0_{d_f\times d_v} & 0_{d\times d_p} \\ 0_{d_f\times d_v} & I_{d_f} & 0_{d\times d_p} \\ 0_{d_v\times d} & 0_{d_f\times d} & 0_{d_p\times d_p} \end{bmatrix} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & x_d \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} = \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0_d \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix}$$

Using the above, the output of the first attention layer added to the residual stream is

$$\begin{aligned} \mathrm{Attn}_1(Z) + Z &= (K_1 Z)(M \odot Z^T Q_1 Z) + Z \\ &= (K_1 Z)(M \odot P_1^T P_1) + Z \\ &= \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0 \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + Z \\ &= \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & x_d + x_{F_1} \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} \end{aligned}$$

Note that W_{11} and W_{21} are identical to W_1 and W_2 in Equation (0), and performing a similar calculation yields

$$\begin{aligned} \mathrm{Block}_1(Z) &= W_{21}\mathrm{ReLU}(W_{11}(\mathrm{Attn}_1(Z) + Z)) + (\mathrm{Attn}_1(Z) + Z) \\ &= \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} = Z_{B_1}. \end{aligned}$$

We denote the output of the first Transformer block by Z_{B_1}.

Step 2: Computing the output of the second block. The second block uses the output of the first Transformer block to compute the second step of the composition. We start similarly by computing the query and value matrices of the attention layer, i.e.,

$$\begin{aligned} &Z_{B_1}^T Q_2 Z_{B_1} = \\ &= \begin{bmatrix} 2x_{F_1}^T & p_{11}^T & p_{21}^T & p_{31}^T \\ 2x_{F_2}^T & p_{12}^T & p_{22}^T & p_{32}^T \\ 2x_{F_3}^T & p_{13}^T & p_{23}^T & p_{33}^T \\ F_1(x_d)^T & p_{14}^T & p_{24}^T & p_{34}^T \end{bmatrix} \begin{bmatrix} 0_{d\times d} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} & 0_{d\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & I_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \\ 0_{\bar{d}_p\times d} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} & 0_{\bar{d}_p\times \bar{d}_p} \end{bmatrix} \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & F_1(x_d) \\ p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \\ &= P_2^T P_2 \end{aligned}$$

and

$$K_2 Z_{B_1} = \frac{1}{2} \begin{bmatrix} 0_{d_v\times d_v} & 0_{d_f\times d_v} & 0_{d\times d_p} \\ 0_{d_f\times d_v} & I_{d_f} & 0_{d\times d_p} \\ 0_{d_v\times d} & 0_{d_f\times d} & 0_{d_p\times d_p} \end{bmatrix} \begin{bmatrix} 2x_{F_1} & 2x_{F_2} & 2x_{F_3} & F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} = \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0_d \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix}.$$

Using the above, we can compute the output of the attention layer in the second Transformer block which evaluates to

$$\begin{aligned}
\mathrm{Attn}_2(Z_{B_1}) + Z_{B_1} &= (K_2 Z_{B_1})(M \odot Z_{B_1}^T Q_2 Z_{B_1}) + Z_{B_1} \\
&= (K_2 Z_{B_1})(M \odot P_2^T P_2) + Z_{B_1} \\
&= \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + Z_{B_1} \\
&= \begin{bmatrix} 3x_{F_1} & 3x_{F_2} & 3x_{F_3} & F_1(x_d) + x_{F_2} \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix}.
\end{aligned}$$

The attention layer uses sub-matrix P_2 of the position encodings to copy the second task token to the data token We repeat the calculations in Equation (0), with W_{21} and W_{22} which yields

$$\begin{aligned}
\mathrm{Block}_2(\mathrm{Block}_1(Z))) &= W_{22}\mathrm{ReLU}(W_{21}(\mathrm{Attn}_2(Z_{B_1}) + Z_{B_1})) + (\mathrm{Attn}_2(Z_{B_1}) + Z_{B_1}) \\
&= \begin{bmatrix} 3x_{F_1} & 3x_{F_2} & 3x_{F_3} & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} = Z_{B_2}.
\end{aligned}$$

Step 3: Computing the output of the final Transformer block. Unsurprisingly, the calculations for the last Transformer block are almost identical. The query matrix is $Z_{B_2}^T Q_3 Z_{B_2} = P_3^T P_3$ and the value matrix is

$$K_3 Z_{B_2} = \frac{1}{3} \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0_d \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix} \begin{bmatrix} 3x_{F_1} & 3x_{F_2} & 3x_{F_3} & F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix} = \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0_d \\ 0_{d_p} & 0_{d_p} & 0_{d_p} & 0_{d_p} \end{bmatrix}.$$

The output of the attention layer in the final block is

$$\begin{aligned}
\mathrm{Attn}_3(Z_{B_3}) + Z_{B_3} &= (K_3 Z_{B_2})(M \odot Z_{B_2}^T Q_2 Z_{B_2}) + Z_{B_2} \\
&= (K_3 Z_{B_1})(M \odot P_3^T P_3) + Z_{B_2} \\
&= \begin{bmatrix} x_{F_1} & x_{F_2} & x_{F_3} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} + Z_{B_2} \\
&= \begin{bmatrix} 4x_{F_1} & 4x_{F_2} & 4x_{F_3} & F_2 \circ F_1(x_d) + x_{F_3} \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix}.
\end{aligned}$$

Passing the output of $\mathrm{Attn}_2(Z_{B_2})$ through the last MLP, yields the output of the Transformer, which is

$$\begin{aligned}
\mathrm{Tr}(Z) &= \mathrm{Block}_3(\mathrm{Block}_2(\mathrm{Block}_1(Z))) \\
&= W_{32}\mathrm{ReLU}(W_{32}(\mathrm{Attn}_3(Z_{B_2}) + Z_{B_2})) + (\mathrm{Attn}_3(Z_{B_2}) + Z_{B_2}) \\
&= \begin{bmatrix} 4x_{F_1} & 4x_{F_2} & 4x_{F_3} & F_3 \circ F_2 \circ F_1(x_d) \\ p_1 & p_2 & p_3 & p_4 \end{bmatrix}.
\end{aligned}$$

Hence, the output of the Transformer is a composition of the three functions F_1, F_2 and F_3 applied to token x_d. □